VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2018 Bc. Branislav Štupák

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

INFORMAČNÍ SYSTÉM PRO PODPORU PRODEJE
INFORMATION SYSTEM FOR SALE SUPPORT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Branislav Štupák**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Macho, Ph.D.**
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**
Ústav automatizace a měřicí techniky

Student: Bc. Branislav Štupák ***ID:*** 158248
Ročník: 2 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Informační systém pro podporu prodeje

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s požadavky na informační systémy pro podporu prodeje a základními legislativními požadavky na zařízení pro evidenci tržeb v České a Slovenské republice a v USA.
2. Proveďte průzkum trhu a rozeberte možnosti realizace informačního systému pro podporu prodeje pro menší firmy. Systém musí umožňovat evidenci tržeb, vytváření statistik a odhady trendů prodeje. Je požadováno, aby klientská část systému byla multiplatformní. Řešte synchronizaci dat mezi klienty prostřednictvím serveru.
3. Navrhněte koncepci systému pro podporu prodeje, vyberte vhodné technologie pro realizaci serverové a klientské části systému.
4. Implementujte klientskou i serverovou část systému a odlaďte je.
5. Sytém otestujte a diskutujte dosažené výsledky.

DOPORUČENÁ LITERATURA:

[1] Šimůnek, M. SQL kompletní kapesní průvodce. 1. dotisk. Praha: Grada, 1999. ISBN80-7169-692-7.

[2] Brázda J. PHP 5 začínáme programovat. Praha: Grada Publishing a.s., 2006. 244 s. ISBN 80-247-1146-X.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 14.5.2018

Vedoucí práce: Ing. Tomáš Macho, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRAKT

Diplomová práca sa zaoberá návrhom a implementáciou systému pre podporu predaja. V prvej časti je vykonaný prieskum trhu a popísané legislatívne požiadávky. Tieto poznatky sú potom uplatnené pri návrhu aplikácie. Návrh systému a jeho požiadávky sú spracované s ohľadom na vysoký užívateľský prežitok(UX). Aplikácia je vytvoréná offline-first prístupom za pomoci frameworku React Native pre cross-platformový vývoj mobilných aplikácií. Pre synchronizáciu dát medzi zariadeniami je použitý databázový systém CouchDB.

KĽÚČOVÉ SLOVÁ

systém pre podporu predaja, miesto predaja, registračná pokladnica, mobilná aplikácia, Android, iOS, cross-platformové riešenie, React Native, Redux, JavaScript, PouchDB, CouchDB, užívateľský prežitok, reštaurácia, predaj

ABSTRACT

The diploma thesis deals with design and implementation of the information system for sale support. In the first part of the thesis is done market analysis and described legislative requirements. These theoretical findings are further applied in the design part of the system. System is designed in spirit of good user experience(UX). Application is created with offline-first approach by React Native framework for cross-platform development of mobile applications. Synchronization between end point devices is done by CouchDB database system.

KEYWORDS

system for sale support, point of sale, cash register, mobile application, Android, iOS, cross-platform solution, React Native, Redux, JavaScript, PouchDB, CouchDB, user experience, restaurant, sale

ŠTUPÁK, Branislav. *Informační systém pro podporu prodeje*. Brno, 2018, 73 s. Diplomová práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedúci práce: Ing. Tomáš Macho, Ph.D.

VYHLÁSENIE

Vyhlasujem, že som svoju diplomovú prácu na tému „Informační systém pro podporu prodeje“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád by som sa poďakoval vedúcemu diplomovej práce pánovi Ing. Tomášovi Machovi Ph.D. za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Brno ....

podpis autora

Obsah

Zoznam obrázkov

Zoznam výpisov

Úvod

Technologický rozvoj dnes umožňuje takmer každému vlastniť mobilný telefón. Množstvo aplikácií a softvérových riešení sa preto presúva na mobilné platformy, aby pomáhali zjednodušovať každodenný život. V niektorých prípadoch je mobilná platforma práve jedinou a postačujúcou platformou pre aplikácie.

Táto práca sa venuje návrhu a realizácii systému pre podporu predaja pre mobilné platformy iOS a Android. Snahou je overiť koncept funkcie takéhoto systému pre podpodu predaja. Výsledným produktom má byť multi-platformová aplikácia, ktorá bude použiteľná pre široké spektrum zákazníkov a nájde si uplatnenie v malých až stredných predajniach a reštauráciach.

V práci bude popísaný návrh a realizácia systému pre podpodu predaja. Návrhu bude predchádzať prieskum trhu, ako aj analýza cieľovej skupiny užívateľov. Následne bude rozobratá legislatívna stránka veci a vyjasnenie požiadávok na takýto systém. Bude popísaný návrh riešenia a následne podrobný návrh jednotlivých častí systému s dôrazom na jednoduché prepoužívanie komponentov. Riešenie má byť zároveň jednoducho ovládateľné, a priniesť zákaznikovi veľký benefit z jeho používania. Z používateľského hľadiska, bude počas celej práce kladený dôraz na kvalitu riešenia a dobrý užívateľský komfort(UX). Následne bude popisaný postup implementácie navrhnutých častí a testovanie celého systému.

1 Prieskum trhu

Na trhu je viacero predajcov registračných pokladní a softvérov pre podporu predaja. Systémom navrhnutým v tejto práci by sme chceli konkurovať bežne dostupným registračným pokladniciam, *predajné miesto – Point of Sale* (POS) systémom a prenosným pokladniciam. Navyše, by malo naše riešenie priniesť kvality a funkcie, ktoré konvenčné pokladne nemajú a to za nižšiu cenu.

1.1 Registračné pokladnice

Firma Elcom patrí medzi najúspešnejšie Európske firmy v oblasti registračných pokladníc a POS riešení. Naším riešením by sme mali konkurovať funkciami napríklad tiež tejto značke, a pri vývoji zohľadniť ich výhody. Firma Elcom ponúka certifikované pokladnice pre rôzne trhy. Jedným z ich najznámejších produktov je registračná pokladnica Elcom Euro-50TE Cash. Jej výhody sú malá veľkosť, prenositeľnosť, trvácne materiály, spĺňa najnovšie legislatívne požiadávky napríklad aj Slovenského trhu.

Obr. 1.1: Registračná pokladnica Elcom Euro-50TE Cash [7]

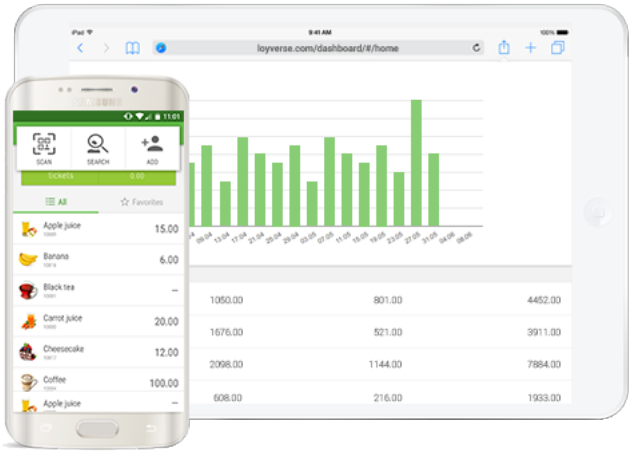
Cena registračnej pokladnice sa pohybuje tesne nad 200€ [7], čo môže byť relatívne vysoká cenovka pre začínajúceho podnikateľa. Za túto cenu dostane zákazník len jednoduché riešenie, často ale postačujúce. Takisto táto pokladnica neponúka žiadne vytváranie štatistík, uchovávanie dát pre neskorší náhľad atď.

1.2 Mobilné aplikácie pre podporu predaja

Na trhu je možné nájsť niekoľko riešení systémov pre podporu predaja pre mobilné platformy, ktoré sa snažia funkciami zabrať svoje miesto v tomto segmente. Viacero

z nich je však nedotiahnutých a nájde sa len málo aplikácií, ktoré sú použiteľné a pokrývajú hlavné príklady použitia pre predaj.

Loyverse aplikácia pre telefóny sa javí počtom stiahnutí ako najúspešnejšie riešenie POS systému na Google Play obchode pre platformu Android. Výrobca ponúka stiahnutie zadarmo a takisto aj jej používanie do istej miery, ďalej je dostupné dokupovanie funkcií. Napriek tomu, aplikácia vyzerá pomerne jednoducho, a aplikácia ponúka funkcie základného predaja a takisto aj jednoduchšie zobrazovanie trendov.



Obr. 1.2: Aplikácia Loyverse POS [8]

Sme názoru, že práve táto aplikácia by mohla byť priamou konkurenciou našej aplikácii a mali by sme našími funkciami prinášať väčší úžitok a lepšie UX aby sme mohli predčiť ich riešenie.

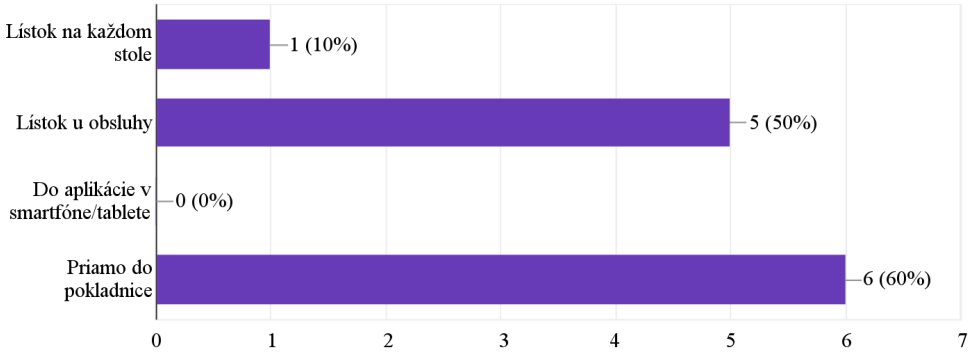
1.3 Analýza cieľovej skupiny

Dá sa očakávať, že systémy pre podporu predaja tu budú ešte mnoho rokov, či už sa jedná o rôzne elektronické pokladnice, alebo softvérové riešenia. Nie je ale vhodné iba na základe tohto tvrdenia stavať vlastné riešeni. Z toho dôvodu je vhodné aby sme pred samotným vývojom aplikácie analyzovali vzorku potenciálnych zákazníkov.

Pre analýzu cieľovej skupiny zákazníkov bolo skriptom stiahnutých okolo 100 mailových adries z webového portálu Zlaté Stránky. [3] Ako vzorka boli vybraté reštaurácie rôzných veľkostí na území Slovenskej Republiky. Pred ich kontaktovaním, je treba vziať v úvahu, že množstvo mailových adries nemusí byť aktívnych, poprípade podnikatelia uvedené mailové adresy ani nepoužívajú. Vytvorili sme krátku reprezentačnú mailovú správu, ktorá v krátkosti predstavila kto sme, aký problém

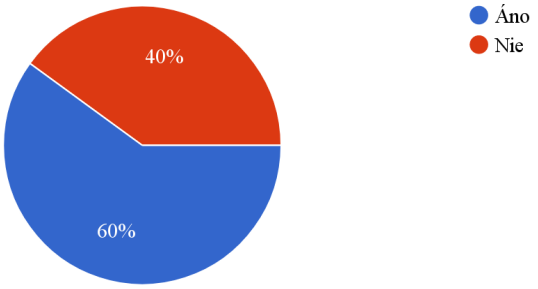
sa snažíme riešiť, pričom v správe boli zákazníci láskavo požiadaný o vyplnenie Google Formulára [4] s možnosťou zanechať svoj kontakt. Za vyplnenie formulára a v prípade zanechania kontaktu bude zákazníkom poskytnutá voľná verzia aplikácie. Daný formulár vyplnilo 10 ľudí, pričom kontakt na seba zanechalo 5 ľudí. Výsledky analýzy boli nasledujúce:

- Akým spôsobom si Vaša obsluha poznamenáva objednávky zákazníkov?



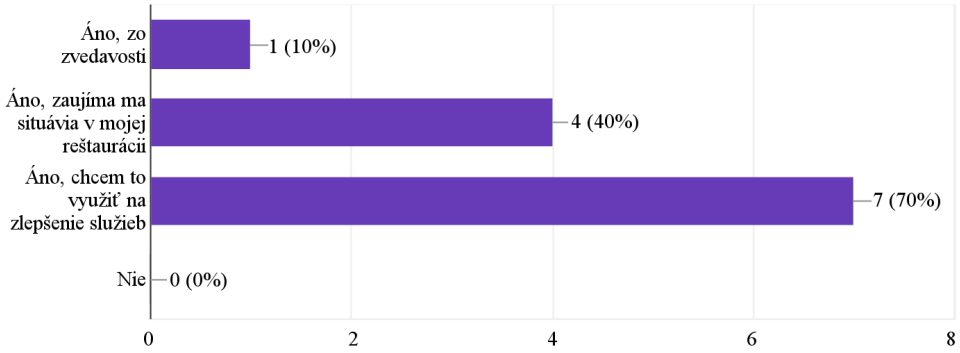
Obr. 1.3: Reakcie potenciálnych užívateľov na otázku č.1

- Využívali by ste aplikáciu v telefóne alebo tablete pre obsluhu vo vašej reštaurácii?



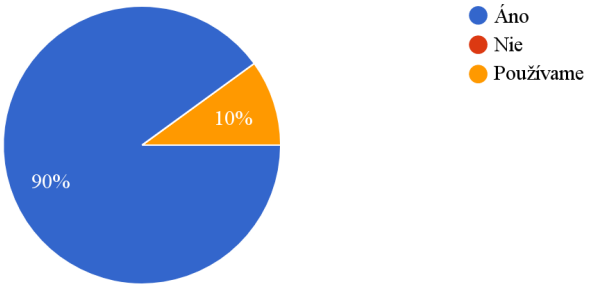
Obr. 1.4: Reakcie potenciálnych užívateľov na otázku č.2

- Zaujímajú Vás štatistiky o Vaších zákazníkoch? (koľko ste mali zákazníkov a kedy prichádzajú, ako dlho u Vás sú, čo si najčastejšie kupujú, koľko peňazí míňajú...)



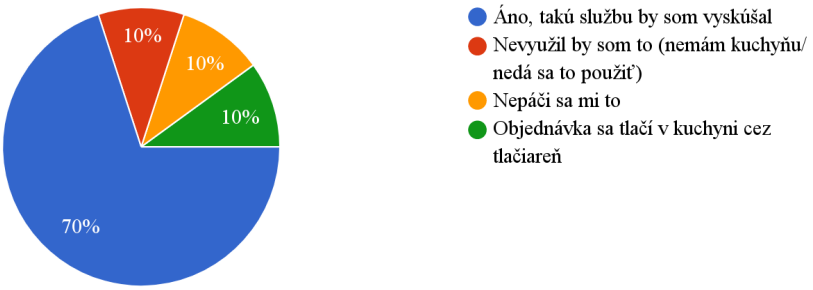
Obr. 1.5: Reakcie potenciálnych užívateľov na otázku č.3

- Používali by ste takú aplikáciu pre obsluhu, ak by Vám ponúkla zaujímavé grafy a štatistiky?



Obr. 1.6: Reakcie potenciálnych užívateľov na otázku č.4

- Obsluha zadá zakázku do tabletu, a tá sa automaticky zobrazí kuchárom na monitore v kuchyni.



Obr. 1.7: Reakcie potenciálnych užívateľov na otázku č.5

Ľudia negatívne reagovali na používanie aplikácie na podporu predaja, no po zmienení výhod ktoré z používania takéhoto systému plynú, je vidieť že boli naklonení myšlienke. Výsledky boli pomerne pozitívne. Na zváženie stojí bilancia úspešnosti analýzy, keďže dotazník vyplnilo 10% celkovo oslovených. Posledná analýza

viac než 2 miliárd marketingových mailov, firmy GetResponse z časového obdobia október-december 2017 vykazuje priemerné 22,88% *ratio otvorenia – Open Rate* (OR) a 4,06% *ratio prekliknutia – Click-Through Rate* (CTR) marketingových mailov. [6] Úspešnosť takýchto mailov sa odvíja od mnohých faktorov ako sú návyky cieľovej skupiny, načasovanie až po obsah marketingového mailu a jeho spracovanie. Úspechom je v našom prípade až vyplmnenie dotazníka, čo vyžaduje nie malú interakciu a čas zákazníka. Odozva na prieskum nás prekvapila, no je treba si uvedomiť že vzorok 10 prevádzok je malý a získané dáta sú iba veľmi orientačné. Analýza nám dala orientačné znalosti, že touto oblasťou sa má zmysel zaoberať.

2 Legislatíva

V tejto časti rozoberieme legislatívu v oblasti používania registračných pokladníc pre Slovenskú republiku, Českú republiku a Spojené štáty americké. Už tu bude vidieť pomerne priepastné rozdiely a nekompatibilitu medzi štátmi. Cieľom tejto časti nieje podrobne rozobrať celú legislatívu zaoberajúcu sa predajom ale povinnosti a náležitosti z toho vyplývajúce. Budeme navrhovať systém, ktorý nebude viazaný na konkrétny štát a jeho zákonné požiadávky. Hlavnou náplňou práce je iba overenie konceptu(POC) systému pre podporu predaja, preto nenavrhneme možno legálny systém. Zváženie zákonných požiadaviek je ale esenciálne pred návrhom a samotnou implementáciou.

Ak chce firma, alebo osoba vytvoriť dôveryhodné riešenie pokladnice pre nejaký trh, čaká ju pomerne zložitá a zodpovedná práca. Legislatíva o používaní pokladnice na prevádzkach je v každej krajine špecifická. Je takmer nereálne navrhnúť systém ktorý by spĺňal legislatívne podmienky všetkých krajín. Taktiež zostaviť systém podľa legislatívnych požiadavok už len jednej krajiny je pomerne zložitá práca. Cieľom tejto práce bude vytvoriť všeobecný systém ktorý nebude prispôsobený pre žiadnu konkrétnu krajinu tak, aby sa na systéme dalo stavať a prispôsobovať ho neskôr pre jednotlivé regióny alebo krajiny.

Javí sa, že používanie aplikácie spolu s tlačiarňou pokladničných bločkov ktorá obsahuje fiskálnu pamäť, by bolo najvýhodnejšie riešenie, keďže väčšina Európskych krajín zo zákona vyžaduje používanie takej pokladnice. Niektoré krajiny ešte stavajú na tomto riešení a vyžadujú online synchronizáciu platieb na štátne servery a ďalšie iné riešenia. Krajiny sveta, ktoré používajú fiskálne pokladničné zariadenia momentálne v sú: Albánsko, Argentína, Bangladesh, Bosna a Hercegovina, Brazília, Bulharsko, Česko, Čile, Chorvátsko, Etiópia, Gambia, Grécko, Kanada, Keňa, Litva, Lotyšsko, Maďarsko, Malta, Montenegro, Rakúsko, Srbsko, Slovensko, Slovinsko, Švédsko, Taliansko, Tanzánia, Panama, Poľsko, Rumúnsko, Rwanda, Venezuela(zoznam je vedený k roku 2017).[9]

2.1 Zariadenie s fiskálnou pamäťou

Zariadenia s fiskálnou pamäťou sú elektronické zariadenia používané na kontrolu daňových príjmov krajiny. Momentálne sú rozšírené vo viacerých krajinách vo svete vrátane Ruska, Bulharska, Srbska, Rumúnska, Macedónska, Albánska, Poľska, Slovenska, Gruzínska, ale tiež v niektorých krajinách Afriky a Ázie. Spomenuté krajiny sú jedné z prvých kde sa pokladne s fiskálnou pamäťou začali používať a v niektorých z nich sú v behu už niekoľko rokov. Postupne sa ale pridávajú ďalšie krajiny, ktoré optimalizujú prijímanie daní a preto sa tento systém teší obľube. V posledných

rokoch boli známe nasadnia v Rakúsku, Českej Republike, Slovinsku atď.[9] Naopak v rozvinutejších serverských krajinách ako Nórsko, Švédsko, Fínsko, alebo tiež aj štáty USA je politika jemnejšia a zákony niesú vôbec tak prísne, krajina má dôveru v ľuďoch a buduje podnikateľské prostredie založené na dôvere.

Fiskálna pamäť je sama o sebe pamäťový modul ako bežná SD karta ktorá je certifikovaná príslušným štátnym orgánom o splnení požiadávok na fiskálnu pamäť. Fiskálna pamäť je zvyčajne šifrovaný modul osadený v zariadení vo forme integrovaného obvodu, pričom táto pamäť je čitateľná len pomocou špecialných prístrojov štátnou autoritou. Fiskálna pamäť zaznamenáva dáta o denných tržbách, pamäť je nezmazateľná a bežným užívateľom nezmeniteľná a nečítateľná.

2.2 Slovenská republika

Podľa zákona č. 289/2008 Z.z. O používaní elektronickej registračnej pokladnice[10] je na území Slovenskej Republiky možné používať:

- Elektronická registračná pokladnica - Elektronické registračné zariadenie vybavené prevádzkovou pamäťou, vstavaným registračným programom, fiskálnou pamäťou, zobrazovacím zariadením pre zákazníka, hodinami, klávesnicou a tlačiarňou ktoré tvoria jeden funkčný celok. Počítač s vlastným registračným programom a so samostatnou fiskálnou tlačiarňou.
- Virtuálna registračná pokladnica - Prostredie zriadené Finančným riaditeľstvom Slovenskej republiky na jeho webovom sídle komunikujúce prostredníctvom koncového zariadenia.

Z vyššie uvedeného vyplýva, že by bolo možné spraviť aplikáciu, ktorá by sa prostredníctvom API pripájala na virtuálnu registračnú pokladnicu Finančného riaditeľstva Slovenskej republiky. Takéto riešenie by bolo najjednoduchšie a s výhľadom do budúcna aj najviac rentabilné a udržateľné. Po kontaktovaní úradu Finančnej správy Slovenskej republiky o sprístupnení API vývojarom a možnosti použitia verejne prístupného API do systému vitruálnej registračnej pokladnice vyšlo najavo, že API k virtuálnej registračnej pokladnici neposkytujú. Virtuálna registračná pokladnica je uzatvorený systém. Napojenie systému virtálnej registračnej pokladnice na systém tretích strán (podnikateľov) nie je možné. Novelou účinnou od 1.1.2018 zákona č. 289/2008 Z. z. o používaní elektronickej registračnej pokladnice a o zmene a doplnení zákona Slovenskej národnej rady č. 511/1992 Zb. o správe daní a poplatkov a o zmenách v sústave územných finančných orgánov v znení neskorších predpisov v znení neskorších predpisov sa jednoznačne ustanoví akým spôsobom a prostredníctvom akých aplikácií je možné prihlásenie do virtálnej registračnej pokladnice, taktiež sa má zaviesť povinnosť používať iba klientské prostredie Finančného riaditeľstva SR, nie externé prostredie pri virtuálnej registračnej pokladnici.

Vzhľadom k stavu štátnych služieb v Slovenskej republike nieje možné takéto riešenie, ako už vyplýva z reakcie Finančnej správy. Ďalšie riešenie je teda postaviť vlastný elektronický systém, ktorý bude využívať fiskálnu tlačiareň. Takéto riešenie je podľa zákona legálne na území Slovenskej republiky. Pre tlač účtovných dokladov potrebuje každá prevádzka tzv. fiskálnu tlačiareň, to jest tlačiaren opatrenú fiskálnym modulom, ktorý slúži na evidovanie jednotlivých platieb pre potreby finančnej kontroly a archivuje údaje o denných tržbách. Vybavenie prevádzky fiskálnou tlačiarňou je zo zákona povinné. Registráciou pokladne na Úrade finančnej správy sa potom rozumie že zariadeniu s fiskálnou pamäťou je priradené jedinečné identifikačné čislo, takže je zaručené, že nebude znova použité pri inom fiskálnom zariadení. Takéto riešenie sa zdá najoptimálnejšie, z pohľadu, že väčšina krajín využíva alebo plánuje využívať spomínané zariadenia s fiskálnym modulom.

2.3 Česká republika

Od roku 2016 dochádza na území Českej republiky k zavádzaniu a optimalizácii zákona o evidencii tržieb. V zákone o evidencii tržieb sa počíta s postupným zavádzaním v niekoľkých termínoch. Najprv evidenciu začali používať reštauračné, stravovacie a ubytovacie služby. Za pár mesiacov nasleduje maloobchod a veľkoobchod a ďalej sa počíta s postupným zavádzaním pre ostatné činnosti(služby, remeselníci atď.). Vláda ČR pristupuje zdá sa k problému obozretne a zavádza optimalizácie a pripomienky podnikateľov do systému.[11]

Systém elektronickej evidencie tržieb (EET) funguje tak, že po prevedenej platbe pokladňa alebo iné elektronické zariadenie odošle informáciu na server Finančnej správy ČR pomocou internetového pripojenia, alebo v teréne pomocou siete GPRS/LTE. Odozva na request je okamžitá odpoveď o potvrdení o zaregistrovaní tranzakcie vo forme unikatného kódu na účtenku. Pokladný systém by mal byť teda schopný vytvoriť XML správu, ktorú podpíše privátnym kľúčom certifikátu vydaného Ministerstvom financií a následne inicializovať šifrovanú SSL komunikáciu so serverom Finančnej správy.

Na rozdiel od riešenia certifikovanej registračnej pokladne so vstavaným neprepisovateľným fiskálnym modulom je toto riešenie pre obchodníkov zdánlivo jednoduchšie a lacnejšie. Je tu ale však potreba brať v úvahu pravidelný poplatok za Internetové pripojenie pokladne, systém tiež kladie značné nároky na spoľahlivosť a priepustnosť dátovej siete a tiež databázový server Finančnej správy, aby platby prebiehali bez problémov. Ďalšou možnosťou pre podnikateľov je používanie webovej aplikácie EET[11], vtedy odpadá nutnosť kúpy pokladnice alebo iného špecialného zariadenia.

Finančná správa Českej republiky sprístupňuje vývojárom a ďalším subjektom verejne dostupné API, technické požiadávky a špecifikácie EET, takže je tak možnosť vytvoriť systém tretej strany ktorý sa napojí do systému elektronickej evidencie tržieb.

2.4 Spojené štáty americké

Spojené štáty americké sú hniezdom úspešných technologických firiem najmä vďaka prostrediu ktoré tam vytvára vláda. Vlády štátov vytvárajú regulácie, ktoré nie sú zložité a podporujú rozvoj a podnikanie. V súčasnosti je pre mnohé firmy americký trh cieľ, kvôli prístupu k nevyčerpateľnému rozpočtu v podobe spoluprác s ostatnými spoločnosťami a prakticky neobmedzeným možnostiam. Ďalšou nespornou výhodou amerického trhu je jeho veľkosť a rozvinutosť. Americký trh ponúka vysoký počet obyvateľov, ktorý je aktuálne viac ako 320 miliónov.[14] Je to trh ktorý hovorí jedným jazykom a je pomerne vzdelaný. Podobne prajná situácia je aj v oblasti legislatívy pokladničných systémov pre stredné a malé biznisy, čo je cieľová skupina tejto práce.

V spojených štátoch amerických sú všetci podnikatelia povinní udržiavať detailný záznam o ich transakciách. Najpopulárnejší spôsob udržiavania záznamov o tržbách je pokladničná kniha, zatiaľ čo je takisto povolené udržiavanie v akýchkoľvek detailných záložkách ktoré obsahujú všetky výdaje a príjmy.[12] Autorita starajúca sa o platby daní a dodržiavanie pravidiel v predaji je známa ako *Služba interných príjmov – Internal Revenue Service* (IRS). Je v záujme podnikateľa, udržiavať prehľadné záznamy o príjmoch a transakciách z podnikania, tak aby bola dostatočná evidencia v prípade auditu zo strany IRS. Kým je zákonom dané udržiavanie záznamov o všetkých transakciách, neexistuje špecifická legislatíva prikazujúca podnikateľom udržiavať záznamy v elektronickej alebo papierovej forme, záznamy musia iba jasne popisovať príjem a výdaje.[13]

3 Požiadávky na systém pre podporu predaja

V tejto časti rozoberieme požiadávky na systém pre podporu predaja. Prejdeme cez časti popisujúce pre koho má byť aplikácia určená, aký problém aplikácia rieši a aký prístup zvolíme pri tvorbe riešenia.

3.1 Užívateľské požiadávky

V tejto časti popíšeme užívateľské požiadávky ako User Story (užívateľský príbeh). User Story má popisovať príbeh z pohľadu užívateľa a vytvárať tak obrázok o riešení. Dajú sa písať rôznymi formami a rôzne upraviť, podľa okolností. My ich budeme písať formou: *Ako <rola> môžem/chcem <funkcionalita>, aby som dostal <benefit>*.

- **US1 Položky** - Ako čašník, chcem mať možnosť upravovať zoznam jedál, aby odpovedali aktuálnemu stavu ktorý máme v ponuke pre zákazníkov.
- **US2 Miesta** - Ako čašník, chcem mať možnosť upravovať zoznam obsluhovaných stolov, aby predstavovali reálny stav v reštaurácii a predaj bol tak vďaka aplikácii prehľadnejší.
- **US3 Predaj** - Ako čašník, chcem predávať položky z ponuky na rôznych stoloch, aby som mal pomocou aplikácie prehľad o zákazníkoch a voľných miestach v reštaurácii.
- **US4 Objednávky** - Ako kuchár, chcem vidieť posledné vytvorené objednávky, aby som mohol v kuchyni pripraviť jedlo, ktoré čašník práve objednal na prevádzke.
- **US5 Detail zákazníka** - Ako čašník, chcem vidieť detaily o zákazníkovi ako sú jeho príchod, počet objednávok a ďalšie, aby som mu vedel poskytnúť lepší servis.
- **US6 Synchronizácia** - Ako čašník, chcem vidieť objednávky ktoré vytvoril môj kolega, aby som mohol pokračovať v servise zákaznikovi, aj keď kolega práve nemôže.
- **US7 Štatistiky predajov** - Ako manažér, chcem mať možnosť vidieť štatistiky o predaji, aby som mohol lepšie riadiť reštauráciu.
- **US8 Štatistiky zamestnancov** - Ako manažér, chcem mať možnosť vidieť štatistiky o svojich podriadených, aby som mohol porovnať ich výkonnosť a podporovať talenty.
- **US9 Vytvorenie prevádzky** - Ako majiteľ reštaurácie, chcem mať možnosť vytvoriť si vlastnú instanciu reštaurácie v aplikácii, aby som mohol používať aplikáciu pre chod prevádzky.

- **US10 Pozvanie do prevádzky** - Ako manažér, chcem mať možnosť pridať zamestnancov do instancie našej reštaurácie, tak aby mohli spolu obsluhovať v reštaurácii pomocou aplikácie a mali prehľad o objednávkach.
- **US11 Doklad o kúpe** - Ako majiteľ reštaurácie, chcem aby zamestnanci mali možnosť tlačiť pokladničné bloky pre zákazníkov, aby sme nemuseli používať ďalší systém tretích strán pre ich vydávanie.

Popísané užívateľské požiadávky sú špecifické pre reštauráciu. Hrajú v nich role *čašník*, *kuchár*, *manažér* a *majiteľ*, tie boli vybrané špecificky, pre lepšiu predstavu o príbehu. Požiadávky sú popísané pre reštauráciu, ale systém budeme navrhovať tak, aby mohol byť použiteľný aj na iných prevádzkach, napr. v rôzných predajniach. Nie všetky požiadávky sú súčasťou tejto práce(tlačenie pokladničných blokov, vytváranie instancií, pozývanie a autentifikácia používateľov atď.), ale pri návrhu ich musíme zohľadniť, keďže sa uvažuje s ich implementáciou v budúcnosti.

3.2 Požiadávky z pohľadu vývojára

Užívatelia

Aplikácia by mala pokrývať širokú škálu prevádzok. Pri tvorbe riešenia budeme klásť dôraz na *užívateľský prežitok – User Experience* (UX) tak, aby riešenie používalo názvoslovie ktoré pokrýva klientov naprieč spektrom predajní, podnikov, reštaurácii atď. Takisto budeme dbať pri návrhu grafiky a obrazoviek na to, aby sme zachovali túto požiadávku.

Online a offline režim

Aplikácia by mala pracovať v režime offline, ale aj online. Pri návrhu aplikácie teda musíme počítať s tým, že nie všetci používatelia majú na mobilných zariadeniach internetové pripojenie. Používateľom musí byť doručený rovnaký UX, a niesmú byť ochudobnení o žiadnú kľúčovú funkciu z aplikácie, ako je napríklad prehľad štatistík a trendov. Naopak v režime online má aplikácia výhodu synchronizácie medzi zariadeniami v čase. To znamená, že napr. v reštaurácii môže každý čašník používať svoje mobilné zariadenie a všetci môžu zároveň obsluhovať akéhokoľvek zákazníka. Tiež navzájom uvidia v reálnom čase vyťaženie reštaurácie, objednávky atď.

Jednoduchosť

Klasické konvenčné registračné pokladne sú pre mnoho používateľov veľmi zložité na úpravu nastavení. Prakticky, každá zmena do listu položiek si v mnohých prípadoch

vyžaduje odborný zásah. Riešením aplikácie chceme ponúknuť jednoduchú úpravu listu položiek.

Multi-platformové riešenie

Úmysel aplikácie je, aby mohla byť používaná na mobilných zariadeniach a tabletoch. V tejto dobe 99,9% trhu mobilných zariadení pokrýva platforma iOS od Apple a Android od Google.[5] Pri tvorbe riešenia budeme dbať na to, aby mohla byť aplikácia používaná na oboch platformách a prinášala používateľom natívny zážitok. Cieľom je tiež neskôr pridať podporu do webového rozhrania pre počítače.

Štatistiky a trendy predaja

Keďže budeme pracovať s dátami predaja jednotlivých prevádzok, ponúka sa možnosť využiť tieto dáta na analýzu. Je to skvelá funkcia, ktorú dnes ponúka len málo firiem. Mnoho predajcov môže ťažiť z toho, že budú mať k dispozícii informácie o tom, čo najviac predávajú, v aký čas prichádzajú ich zákazníci, a mnoho ďalších. Táto funkcia by mala byť podporovaná ako v režime online, tak v režime offline. Pri tvorbe tejto funkcie budeme dbať na to, aby mohla byť jednoducho prepoužitá znova vo webovom rozhraní.

Cena

Aby aplikácia mohla dosiahnúť väčší úspech, je vhodné aby bola zadarmo a ponúkala užívateľom skúsiť si jej funkcie, aj keď v obmedzenom čase. Je mnohými analýzami dokázané, že práve model nakupovania v aplikácii (In-app purchase) je dnes najvýnosnejší model predaja aplikácií.

Prispôsobenie aplikácie a balíčky

To, že máme možnosť prevádzkovať aplikáciu na platforme iOS a Android, nám dáva veľké možnosti ako využívanie senzorov, internetového pripojenia a dostatočný výkon zariadenia. Aplikáciu budeme budovať s ohľadom na to, že by mohli byť do nej pridávané monetizovateľné balíčky, ktoré by užívateľom pridávali ďalšie možnosti. Príkladom môže byť napr. balíček podpory pre daný trh, podpora kryptomien pre platbu, podpora debetných kariet, platobného terminálu a ďalšie.

Edukácia

V dnešnej dobe je možné sledovať zvyšujúce sa povedomie podnikateľov o marketingu a začínajú sa viac zaujímať o svojho zákazníka, čo im samozrejme zlepšuje

tržby. V časti aplikácie, kde budú uvedené štatistiky a trendy predaja by bolo vhodné pridať užívateľom informácie ako s týmito dátami môžu nakladať, krátke tipy na zlepšenie imidžu a marketingu prevádzky.

3.3 Metodika vývoja

Do nedávna obľúbené vodopádové(waterfall) metódy vývoja vykazovali viacero nedostatkov. Za ten najväčší môžme považovať neschopnosť sa dynamicky adaptovať zmenám požiadaviek počas vývojového cyklu. Ak sa aj takáto zmena prijala, jej nasadenie je z povahy tohto modelu ťažkopádne a neočakávané. V praxi vývoj touto metodikou prebieha nasledovne: Nastaví sa 3 mesačný cieľ, vypracuje sa zadanie požiadávok a následne sa po dobu 3 mesiacov pracuje na riešení. Z príkladu je jasné, že hneď na začiatku sa očakáva pomerne presný popis čo chceme dosiahnúť a presné požiadávky jednotlivých častí systému. Odhadnúť s takým veľkým predstihom všetky požiadávky a možné nárazové zóny je veľmi náročné. Dovolíme si tvrdiť že táto metodika zabila mnoho potenciálne dobrých nápadov a projektov. Z povahy vodopádovej metodiky vývoja softvéru vyplýva viacero nedostatkov:

- Zákazník dostáva výsledný produkt po dlhej dobe.
- Zákazník nie je schopný dávať spätnú väzbu počas vývojového cyklu.
- Vývojár obvykle po dlhom vývojovom cykle dostane spätnú väzbu od zákazníka, ktorý nedostal softvér podľa jeho predstáv.
- Neprebieha kontinuálna komunikácia medzi stranami a nevybuduje sa tak dobrý vzťah.
- Zákazník dostáva veľké zmeny na ktoré je ťažké reagovať pozitívne.

Agilné metodiky

Nedostatky vodopádových metodík sa dajú previesť aj do startupového sveta. V prípade že zadávateľ má skvelý nápad na realizáciu(na začiatku sa zdajú byť skvelé všetky nápady). Následne tým pracuje na riešení rok a až po tejto dobe sa púšťa riešenie do sveta, nastávajú situácie kde zadávateľ zisťuje, že o produkt nie je záujem, nápad nie je až taký skvelý a riešenie je ťažkopádne na adaptáciu pre reálny svet a potenciálneho používateľa.

Práve naopak, sa ukázalo že dodržiavanie agilných metodík dokonca aj v tomto smere prináša väčšiu úspešnosť. V prípade dodržiavania akejkoľvek metodiky je na začiatku dôležité overenie trhu, či je o daný produkt záujem. Dostatočným riešením je oslovenie vzorky potenciálnych zákazníkov s vyplnením dotazníka. Za výmenu za neskorú licenciu alebo otestovanie produktu vám potenciálni zákazníci radi pomôžu.

Takáto analýta pomáha vyformovať počiatočnú myšlienku a pridať ďalšie časti riešenia poprípade odložiť také ktoré niesú zaujímavé pre zákazníka. Následne je dôležité čo najskôr priniesť základnú verziu aplikácie do reálneho sveta. Nie je dôležité aby riešenie bolo perfektné, práve naopak má ukazovať základnú myšlienku, a problém ktorý rieší. Tak sa môže ďalej zisťovať o čo majú zákazníci záujem a kam má ďalej smerovať vývoj produktu. Pripomína to agilný vývoj.

Pri agilnom vývoji sa využívajú rôzne agilné metodiky. Sú to skupiny metód, ktoré nastavujú proces vývoja softvéru tak aby bol iteratívny a inkrementálny. Obvykle je to jednoduchá skupina pravidiel riadenia tímu a vývoja softvéru. Dodržiavanie daných pravidiel umožňuje rýchly vývoj softvéru a zároveň dokážeme reagovať na zmenu požiadaviek počas vývojového cyklu. Podľa týchto metodík sa správnosť systému overí jedine pomocou rýchleho vývoja, predložení riešenia zákazníkovi a následných úprav podľa spätnej väzby. Ako bolo spomenuté vyššie agilný vývoj nie je obmedzený iba na programovanie, ale našiel svoje uplatnenie takisto v iných oblastiach. Dá sa povedať, že protikladom agilného prístupu je waterfall model.

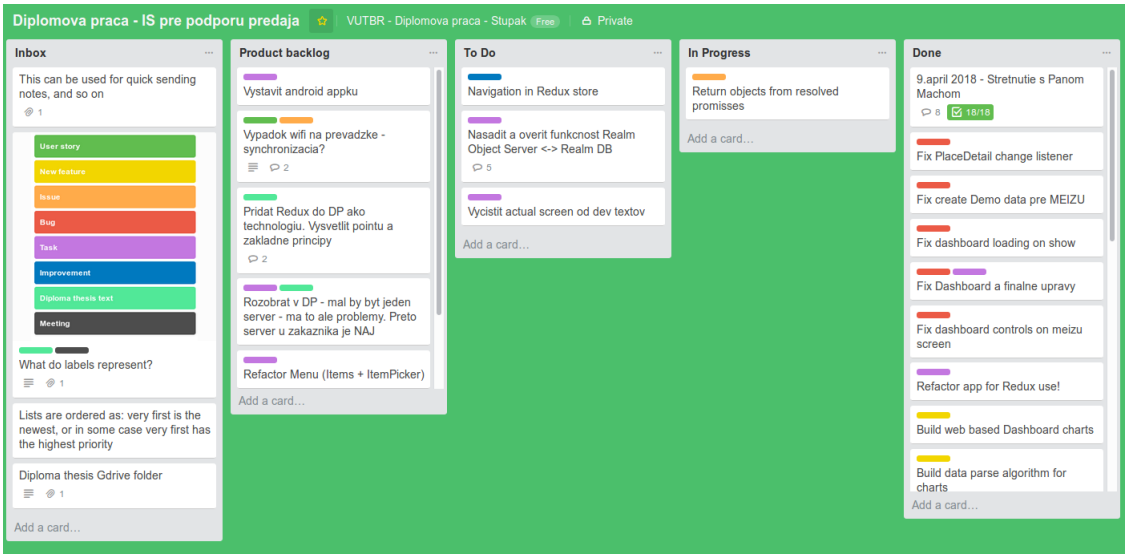
Techniky používané agilnými metódami sa často používali už skôr, ale pojem sa začal používať až po roku 2001. V Utahu sa vtedy zišli odborníci z oblasti softvérového inžinierstva a vývoja softvéru, kde zadefinovali Manifest agilného programovania[1], ktorý pozostáva z nasledujúcich bodov:

- Našou najväčšou prioritou je vyhovieť zákazníkovi častým a priebežným dodávaním hodnotného softvéru.
- Vítame zmeny v požiadavkách, a to aj v neskorších fázach vývoja. Agilné procesy podporujú zmeny vedúce ku zvýšeniu konkurencieschopnosti zákazníka.
- Dodávame fungujúci softvér v intervaloch týždňov až mesiacov s preferenciou kratšej periódy.
- Ľudia z biznisu a vývoja musia spolupracovať denne po celú dobu projektu.
- Budujeme projekty okolo motivovaných jednotlivcov. Vytvárame im prostredie, podporujeme ich potreby a dôverujeme, že odvedú dobrú prácu.
- Najúčinnejším a najefektívnejším spôsobom zdieľania informácií vývojovému tímu z vnútra aj z vonku je osobná konverzácia.
- Hlavným merítkom pokroku je fungujúci softvér.
- Agilné procesy podporujú udržateľný rozvoj. Sponzori, vývojari aj užívatelia by mali byť schopní udržať stále tempo dlhodobo.
- Agilitu zvyšuje neustála pozornosť venovaná technickej výnimočnosti a dobrému dizajnu.
- Jednoduchosť umenia maximalizovať množstvo nevykonanej práce je kľúčová.
- Najlepšie architektúry, požiadávky a návrhy výjdu zo samo-organizujúcich sa tímov
- Tím sa pravidelne zamýšľa nad tým, ako sa stať efektívnejším, a následne

koriguje a prispôsobuje svoje správanie a zvyky.

Nástroje

Pre vývoj nášho systému je vhodné zvoliť metodiku podobnú agilnej. Nie je potreba prísne dodržiavať niektorú obdobu agilného systému, no považujeme za vhodné sa tým inšpirovať. V projekte je zákazník vedúci práce, resp. študent. A vývojár je študent. Ten reportuje vedúcemu práce priebeh v pravidelných týždenných intervaloch. Pre vývoj bude použitý nástroj Trello, kde boli vytvorené 4 stupne úloh. Zvolili sme kategórie *Product backlog*, *To Do*, *In Progress* a *Done*. Kategórie jasne popisujú zaradzovanie úloh a všetci zúčastnení v projekte majú prehľad o tom, na čom sa aktuálne pracuje a aké sú ich aktuálne úlohy. Používanie nástroja v spojení s našou upravenou agilnou metodikou výrazne uľahčí vývoj a robí ho tranparentným.

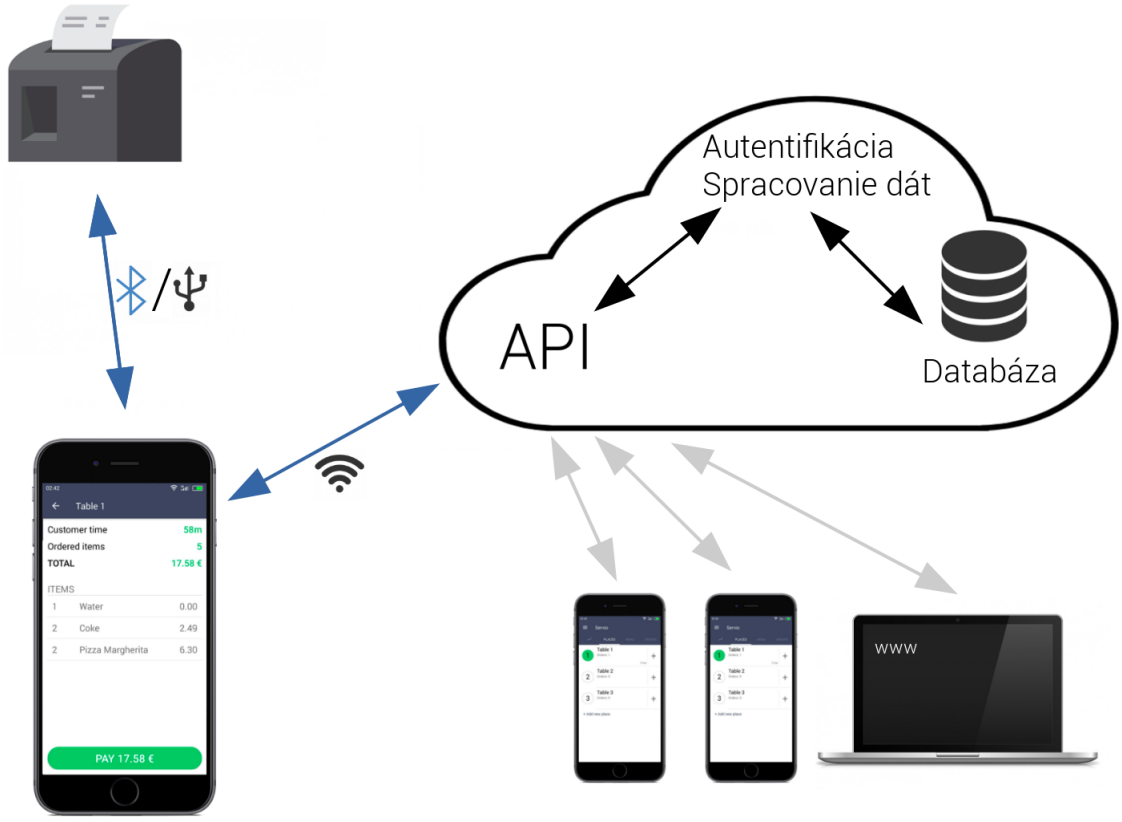


Obr. 3.1: Aplikácia Trello používaná počas vývoja

4 Návrh koncepcie systému

Zmyslom tejto práce je overiť koncept(POC) riešenia systému pre podporu predaja na mobilných platformách. Síce z práce nevznikne systém vhodný okamžitého nasadenia do prevádzky, budeme smerovať všetky kroky tak, aby sa tomu stavu výsledok priblížil. Systém aktuálne nenavrhujeme pre špecifický trh, radšej ho navrhujeme spôsobom, aby mohol byť neskôr jednoducho nastaviteľný pre jednotlivé trhy.

4.1 Koncept riešenia



Obr. 4.1: Koncept systému pre podporu predaja

Z požiadaviek na aplikáciu vyplýva, že riešenie by malo bežať na mobilných zariadeniach a tabletoch. Práve tam chceme cieliť a snažíme sa priniesť používateľom lacné riešenie s funkciami pokročilého systému pre podporu predaja.

Aplikáciu budeme budovať tzv. offline-first prístupom. Chceme, aby každá samostatná instancia aplikácie dokázala bežať za akýchkoľvek podmienok. To znamená, že zariadenie musí byť schopné napríklad ukazovať štatistiky, vytvárať objednávky a spĺňať ďalšie funkcie aj bez prístupu k Internetu. Z toho dôvodu bude aplikácia obsahovať databázu, ktorá sa bude synchronizovať s databázou na serveri.

Všetky funkcie musia byť podporované nezávisle na platforme(iOS alebo Android). Zobrazovanie štatistík na mobilnom zariadení bude plne offline, čo pomôže k rozloženiu zaťaženia zo serverovej časti. Časť štatistík by mala vedieť bežať vo webovom prehliadači, aby mohla byť neskôr jednoducho prepoužitá.

Riešenie backendu na serveri vytvorí styčný bod, cez ktorý sa budú dáta medzi jednotlivými zariadeniami synchronizovať. Takisto nám to umožní neskôr pridať webové rozhranie pre aplikáciu, napríklad prezeranie štatistík na počítači, alebo aj ďalšie funkcie. Serverová strana bude tiež obsahovať databázu, ktorá by mala obsahovať tie samé dáta ako mobilné koncové zariadnia. Veľkou výhodou tohto riešenia bude, že ak kedykoľvek pozveme nového užívateľa s novým zariadením do našej instancie prevádzky, stiahnú sa mu aktuálne dáta. Serverová časť zároveň slúži na zálohovanie dát a možnosť zobrazovať štatistiky na počítači.

Je nutné podotknúť prípad výpadku internetového spojenia, alebo pobočiek bez internetového pripojenia. Z toho dôvodu serverová časť musí byť navrhnutá tak, aby mohla bežať aj na samostatnom serveri u zákazníka. Vhodné by bolo použiť lacný mini počítač prevádzkujúci operačný systém Linux ako napr. Raspberry Pi alebo jeho klony, čo by výrazne znížilo cenu oproti klasickému serverovému riešeniu. Prístup by sa potom riešil, tak že sa vytvorí WiFi AP na ktorý sa pripoja používatelia. Výhodou je možnosť poskytovania internetového pripojenia, vrámci toho samého AP a to celkom jednoducho premostením lokálnej počitáčovej siete (LAN) a WiFi v rámci operačného systému.

Aplikácia by mala byť schopná v prípade online režimu aktualizovať údaje o prevádzke v reálnom čase na všetkých pripojených zariadeniach. Ak časník na svojom zariadení objedná niečo na stole X, kuchár v kuchyni by mal vidieť, že nám pribudla nová objednávka a takisto aj ostatní čašníci by mali túto zmenu vidieť na svojích zariadeniach. Z toho vyplýva, že serverová časť musí umožnovať synchronizáciu v reálnom čase medzi zariadeniami. V prípade, že je zariadenie nejakú dobu offline, po pripojení si musí natiahnúť všetky dáta tak, aby boli na všetkých zariadeniach na prevádzke rovnaké a aktuálne.

Mobilné zariadenie na ktorom bude systém pre podporu predaja bežať, by malo byť schopné spustiť tlač účteniek z externého zariadenia. Je možné k aplikácii ponúkať aj zariadenie, pripojiteľné cez rozhranie Bluetooth, alebo USB. Tlačiareň pokladničných blokov by mala byť opatrená fiskálnym modulom, aby spĺňala zákonné podmienky viacerých krajín, nie je to však podmienka napr. pre americký trh. Tlačiarne účteniek a pokladnice používajú štandardizovaný ESC/POS protokol, takže ak bude overená funkčnosť na jednom zariadení, nemal by byť problém funkciu prevádzkovať aj na ďalších rôzných zariadeniach. Táto časť však nie je súčasťou zadania práce, je ale potrebné pri návrhu na to myslieť.

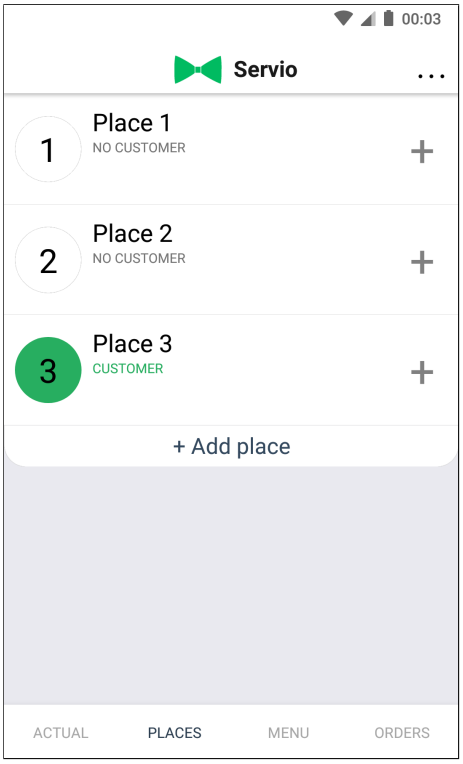
Aplikácia by mala neskôr dostať aj webové rozhranie, primárne kvôli prezeraniu

štatistík a trendov, keďže je používanie počítača príjemnejšie spojené s manažérskou prácou. Webové rozhranie štatistík nie je súčasťou tejto práce.

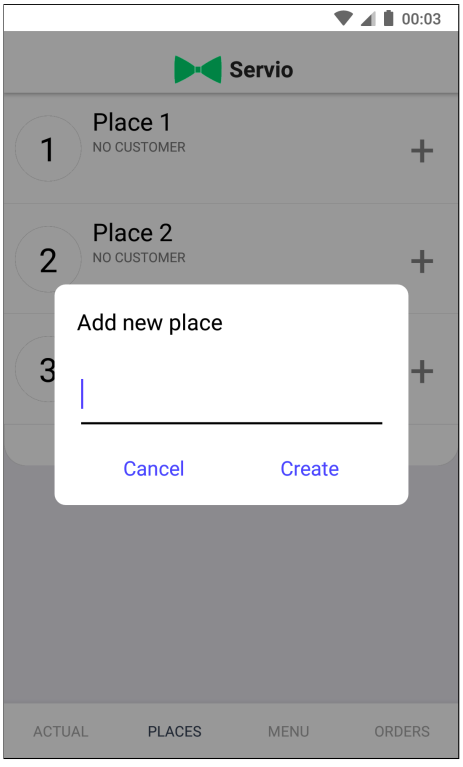
4.2 Návrh obrazoviek klientskej časti

Obrazovky sa snažíme navrhnúť tak, aby bolo ovládanie aplikácie jednoduché a intuitívne. Aplikáciu navrhujeme pre platformy iOS a Android, ktoré majú špecifické odporúčania na dizajn, tak aby používateľ mal pocit že aplikácia ladí so systémom. Predmetom tejto práce ale nie je navrhovať dva rozdielne dizajny, ale radšej nájsť kompromis medzi nimi a znížiť tak cenu vývoja aplikácie.

Dizajn aplikácie bude postavený na kartách, ktoré budú jednoducho prepínateľné z lišty, alebo potiahnutím obrazovky v horizontálnom smere. Na lište má užívateľ možnosť prepínať sa medzi kartami *Aktálne*, *Miesta*, *Menu* a *Objednávky*. Obrazovka *Aktálne* nie je časťou tejto práce, preto sa ňou nebudeme zaoberať. Na obrázkoch 4.2 a 4.3 je vidieť obrazovku zobrazujúcu miesta a dialóg pre ich vytváranie. Dialógy budú prepoužívané, to znamená, že pre vytvorenie alebo úpravu sa použije ten istý vzhľad(taktiež tá istá komponenta) iba so zmenenou textáciou a akciami pri kliknutí.



Obr. 4.2: Obrazovka miesta

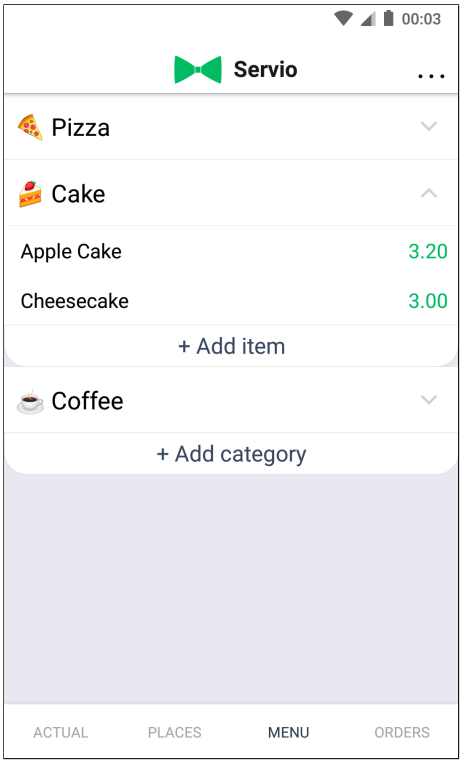


Obr. 4.3: Pridanie a úprava miesta

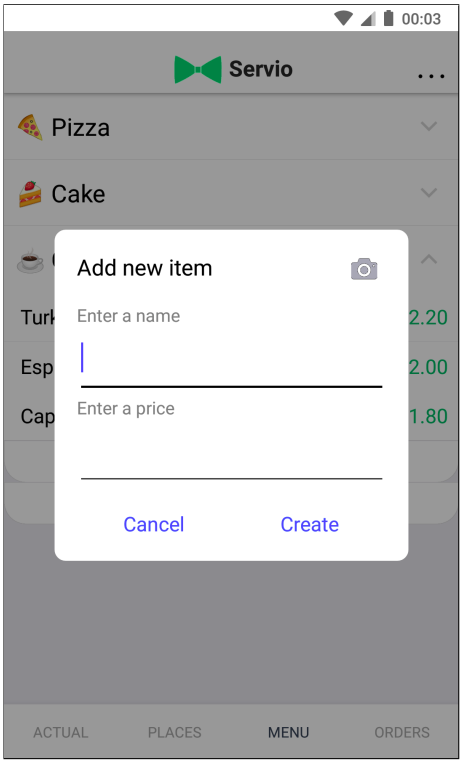
Obrazovka miesta umožňuje rýchlym pohľadom zistiť aktuálny stav na prevádzke. Bolo by vhodné pridať viac detailov o zákaznikovi, to však ukáže spätná väzba

od používateľov. Kliknutím na *plus* ikonu pri mieste, užívateľa prevedieme na obrazovku 4.9 pre vytvorenie objednávky. Po vytvorení objednávky užívateľa prevedieme naspäť na obrazovku na ktorej začínal, pričom tento pristup volíme vrámci celej aplikácie pre zachovanie jednotného dojmu a dobrého UX. Kliknutie na *+Add place* otvorí dialógové okno na obrázku 4.3 vpravo, pre pridanie nového miesta.

Na obrázkoch nižšie je ukážka spravovania položiek pre predaj. Zoznam predajných položiek je jednou z kariet hlavnej obrazovky, na vybranie zo spodnej lišty. Očakávame, že užívateľ bude mať záujem položky organizovať v rámci kategórií. Pre zobrazenie bol zvolený rozbaľovací list tak ako ukazuje obrázok 4.4. V najvyššej úrovni je tlačidlo pre pridanie kategórie a pri vnorení sa do kategórie zase tlačidlo pre pridanie predajnej položky. Dialóg pre pridanie položky je na obrázku 4.5, ktorý je opäť jednoducho prepoužiteľný aj pre úpravu položiek. Dialóg by mohol obsahovať možnosť odfotiť položku, pretože potom pri vytváraní objednávky je jednoduchšie pracovať s obrázkami, než čítať názvy položiek. Túto možnosť by bolo možné pridať v ďalšej verzii aplikácie. Tiež by bolo vhodné, aby v prípade keď užívateľ neodfotí položku, môžme na základe názvu položky vygenerovať farbu, ktorá by bola použitá namiesto fotografie, zachovali by sme tak tento jednoduchý koncept.



Obr. 4.4: Obrazovka položky

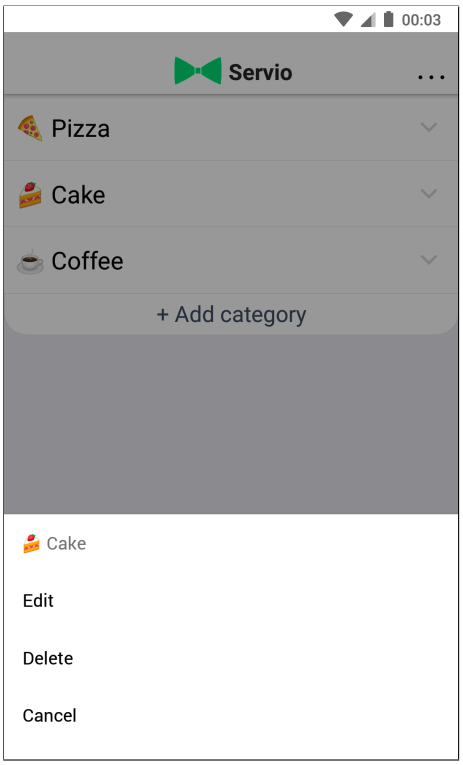


Obr. 4.5: Pridanie a úprava položky

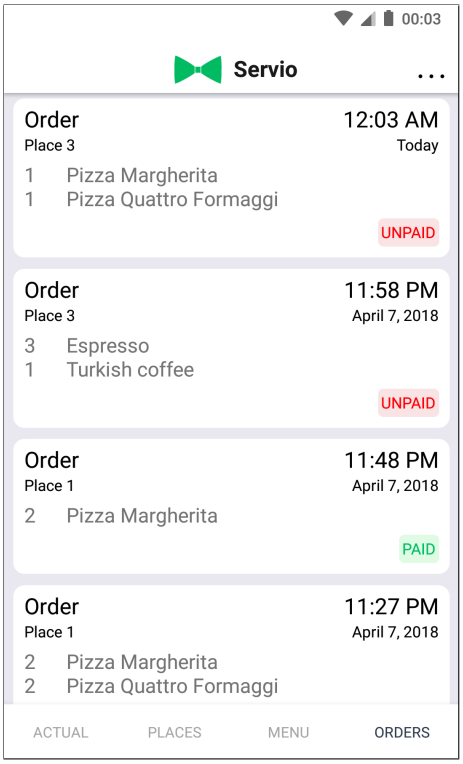
Je potrebné dovoliť užívateľovi upravovať dáta, z toho dôvodu použijeme lištu akcií (actionsheet), ktorá je uvedená na obrázku 4.5. Toto menu bude vyvolané dl-

hým stlačením položky v liste. Lištu budeme implementovať pre miesta a položky. Neskôr by bolo vhodné ju naimplementovať aj pre objednávky, na teraz v jej implementácii do objednávok nie je dôvod. Lišta bude obsahovať možnosti pre úpravu, zmazanie a zrušenie akcie, v jej záhlaví by mala byť zadefinovaná položka ktorú sme vybrali.

Zobrazenie objednávok pre celú pobočku je veľmi dôležité. Obrázok 4.7 ukazuje návrh karty objednávok. Do tohto zoznamu sa budú zahŕňať objednávky vytvorené naprieč všetkými miestami v rámci instancie našej pobočky. Zoznam má plniť úlohu v kuchyni v reštaurácii, alebo v sklade predajne atď. Položky v zozname budú zoradené od najnovšej objednávky až po najstaršiu vytvorenú. Objednávka v zozname musí ukazovať základné informácie o objednávke a to zoznam položiek objednávky, miesto kde bola vytvorená, čas vytvorenia a jej stav. Je možné pridať ďalšie funkcie ako dlhé kliknutie, alebo preklik z objednávky na miesto ku ktorému patrí. Tieto funkcie ostanú na ďalšie zváženie, nie sú zahrnuté v aktuálnom návrhu.



Obr. 4.6: Lišta akcií

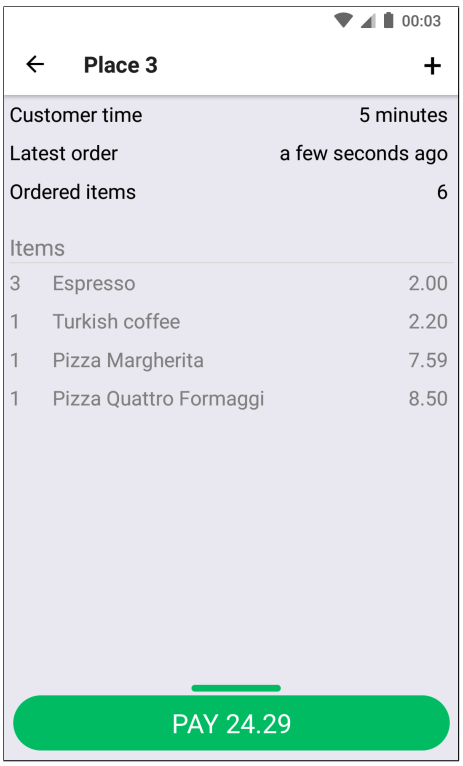


Obr. 4.7: Obrazovka objednávky

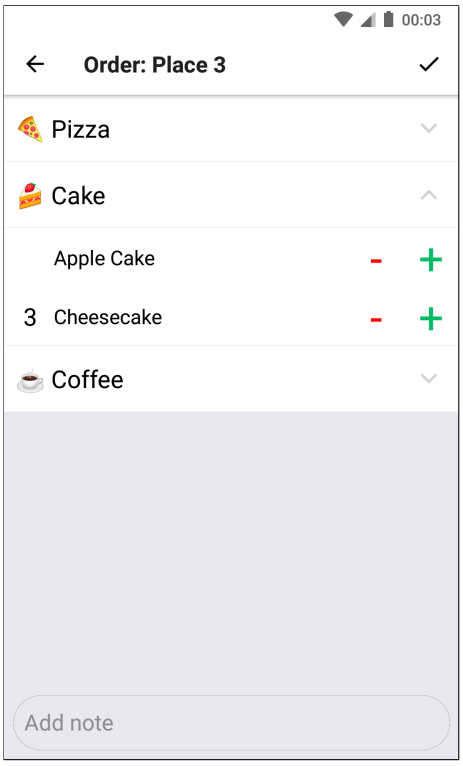
Pri kliknutí na miesto na obrazovke 4.2 sa vyvolá zobrazenie detailu miesta. V prípade že tam aktuálne nie je žiaden zákazník, sa zobrazí iba jednoduchá obrazovka s možnosťou vytvorenia objednávky. V opačnom prípade sa zobrazí detail miesta 4.8. Z tohto pohľadu je možné spravovať zákazníka a získať prehľad o ňom a jeho objednávkach. V pravom hornom rohu je možné vytvoriť novú objednávku, pričom

po jej vytvorení bude užívateľ presmerovaný späť na detail. Na spodnej strane je tlačidlo pre zaplatenie všetkých objednávok zákazníka. Aktuálny návrh počíta s jednoduchým zaplatením, no neskôr by bolo vhodné pridať ďalšie možnosti platenia ktoré by riešili čiastočnú platbu, platbu s voľbou tlačenia dokladu a iné. Potiahnutím tlačidla pre platenie hore, by sa zobrazilo rozšírené menu.

Na obrázku 4.9 vpravo je ukážka vytvárania objednávky, ktorá prepoužíva vzhľad obrazovky 4.4. Užívateľ má možnosť intuitívne navoliť počet vybraných položiek pre objednávku, alebo tiež pridať špecifickú poznámku k objednávke. Poznámka je potom zobrazená v detaile zoznamu na obrazovke *Objednávky*. Nevýhodou takéhoto zobrazenia je zložitosť pri používaní a hodilo by sa jednoduchšie zobrazenie, napr. použitie miniatúr. Pri vytváraní predajných položiek (obrázok 4.5) bolo počítané s možnosťou do budúcna pre pridanie fotografie, alebo generovania farby pre položky. Takéto zobrazenie by mohlo podstatne zjednodušiť používanie systému.



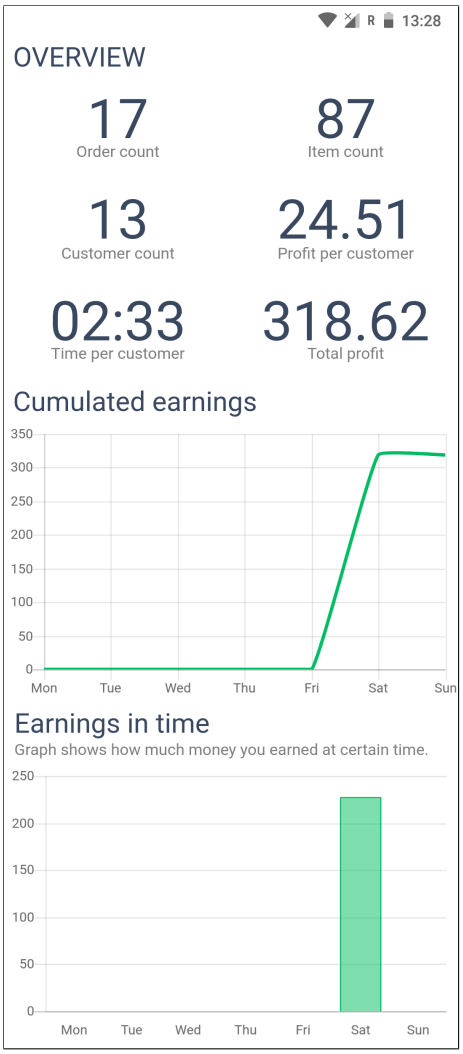
Obr. 4.8: Detail miesta



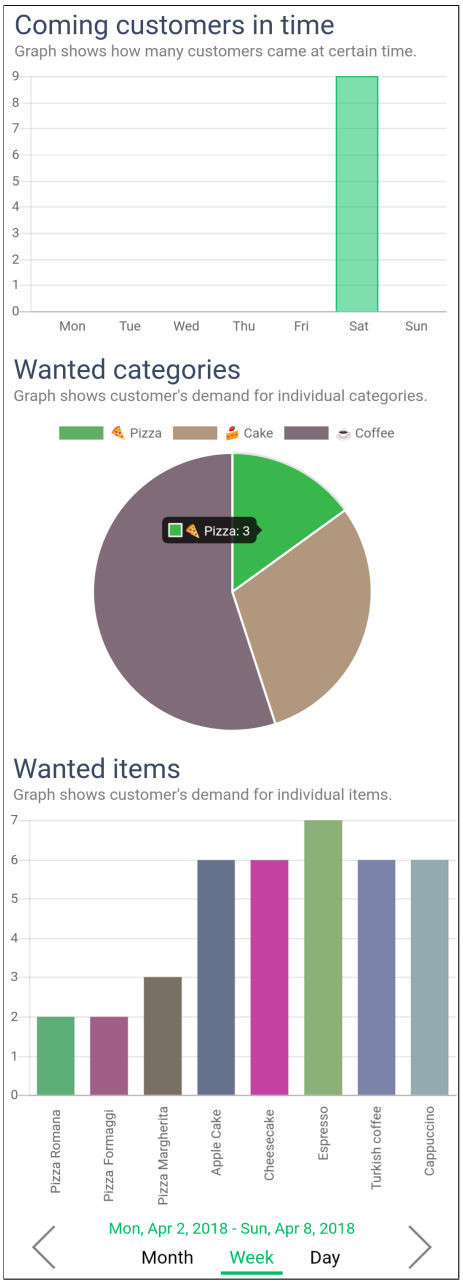
Obr. 4.9: Vytvorenie objednávky

Súčasťou klientskej časti aplikácie bude aj zobrazovanie štatistík o predaji. Na obrázku 4.10 je uvedený možný návrh obrazovky štatistík, ktorý pokračuje na obrázku 4.11. Na spodnom okraji obrazovky zariadenia je panel pre ovládanie časového rozsahu, ten bude umožňovať prepínanie medzi rozsahmi *deň*, *týždeň* a *mesiac*. Na okrajoch panelu by sa mali nachádzať ovládacie prvky pre posun v čase. Obrazovka sa môže rozširovať o ďalšie funkčné grafy, alebo štatistiky, pričom stále by sa veľ-

kosť prvkov mala prispôsobiť tak, aby sa vošli vertikálne do zobrazenia na zariadení. Vhodné by bolo tiež pridať nejaký spôsob stránkovania funkčných celkov štatistík, tak že by užívateľ pri potiahnutí videl zarovnaný funkčný celok. V tomto štádiu však postačuje, že sa budeme po obrazovke posúvať vertikálne skrolovaním, bez stránkovania.



Obr. 4.10: Obrazovka štatistiky 1



Obr. 4.11: Obrazovka štatistiky 2

5 Výber vhodných technológií

V tejto časti je popísaný výber vhodných technológii pre realizáciu systému. Pri výbere technológií berieme ohľad na platformy na ktoré chceme klientskú aplikáciu cieliť a to sú platformy iOS a Android. Takisto je treba zvážiť náročnosť projektu a vzhľadom k tomu zvoliť vhodné technológie.

5.1 Klientská časť systému

5.1.1 Natívne technológie

Vývoj natívnych apikácii prináša so sebou viacero výhod, no aj jednu veľkú nevýhodu a to je cena vývoja. V prípade že je vyžadovaný vývoj aplikácie pre platformu iOS aj Android, je potrebné v podstate aplikáciu programovať dvakrát, keďže obe platformy primárne používajú odlišné technológie. Výhodami samozrejme je vysoký výkon aplikácií a používanie natívnych komponentov pre vývoj. Využívanie natívnych komponentov prináša tiež natívny UX pre používateľa.

Pre platfromu iOS sa dlhú dobu používal pre natívny vývoj programovací jazyk Objective-C, no v roku 2004 firma Apple ohásila zmenu primárneho programovacieho jazyka na modernejší jazyk Swift.

Pre platformu Android sa od jej počiatku ako primárny jazyk pre vývoj používala Java pre Android. Jazyk nesie všetky prvky klasickej Javy a má pridané ďalšie rozšírenia Androidu v rámci jeho SDK. V roku 2017 firma Google ohlásila, že novým primárnym jazykom pre vývoj aplikácií sa stane jazyk Kotlin. Ten je plne kompatibilný s Javou a dokáže využívať všetky štruktúry Javy. Jeho veľkou výhodou je tzv. syntactic sugar, ktorý vo veľkej miere uľahčuje vývojárom prácu, tiež práca s ním je omnoho príjemnejšia v porovnaní s Javou. Kotlin sa tiež kompiluje do Java bytecodu a rovnako bežia všetky aplikácie v ňom napísane na JVM.

Vyššie spomenuté technológie sa dnes v plnej miere používajú pre vývoj natívnych mobilných aplikácií. V tejto práci by sme sa chceli zamerať na vývoj cross-platformovej aplikácie. Už pri jej zrode teda počítame s tým, že by mala bežať minimálne na oboch platformách. Preto je podľa môjho názoru vývoj dvoch natívnych aplikácií nie vhodný, keďže sa snažíme zachovať nízku cenu riešenia.

5.1.2 Cross-platformové aplikácie

Ako bolo spomenuté v predchádzajúcej časti, natívne aplikácie sú vyvýjané pre každý operačný systém osobitne s použitím daného *softvérový vývojový balíček – Software Development Kit* (SDK). Existujú však technológie ktoré umožňujú písať

cross-platformové natívne aplikácie. V princípe fungujú tak, že sa napíše kód v nejakom jazyku a podľa neho sa potom vygeneruje natívny kód pre danú platformu. Takým spôsobom funguje napríklad framework Xamarín ktorý sa píše v jazyku C#. Podobne funguje aj framework React Native, ten nám umožňuje napísať aplikáciu pomocou Javascriptu, pričom využívame javascriptové komponenty, ktoré reprezentujú natívne komponenty. Takto vytvorené aplikácie majú výhodu práve v tom, že sa aplikácia píše iba raz, a tiež je vývoj väčšinou uľahčený tým, že sa píše vo vyššion a príjemnejšom jazyku a nie je potrebné často riešiť zložité fundamentálne problémy ako pri klasickom vývoji natívnych aplikácií.

Ďalším spôsobom je písanie hybridných HTML aplikácií, keďže obidve platfromy iOS aj Android majú komponentu webového okna - WebView. Je teda bežnou praxou písať aplikáciu pomocou webových technológií HTML, CSS a Javascript a ďalších a potom túto aplikáciu púšťať v natívnej komponente WebView. Takto vytvorené aplikácie majú obrovskú výhodu v cene. Riešenie sa dá potom prepoužiť aj pre webové rozhranie, čo je ďalšia výhoda. Nevýhodou riešenia ale je nedostatočný výkon, aplikácie sú pomalé a je zložité docieliť natívny UX pre užívateľa. Z toho dôvodu sa od týchto riešení v súčasnej dobe upúšťa.

5.1.3 React Native a vývoj cross-platformových aplikácii

Open-source framework React Native od firmy Facebook sa pomaly stáva veľkým hráčom vo vývoji mobilných aplikácií. Stále viac zákazníkov požaduje vývoj v tejto technológií, kvôli mnohým pozitívam. Vývoj v ňom je lacnejší pričom sa vyvýjame stále natívnu aplikáciu s vysokým výkonom. React Native prevzal výhody webového frameworku React a implementuje ich do mobilných aplikácií, oproti Reactu obsahuje naviac natívne komponenty.[16] Prakticky teda vývojár ktorý pozná React dokáže používať React Native, a naopak. Ďalšie výhody frameworku sú:

- React Native používa viac deklaratívny prístup ako imperatívny prístup ako mnohé iné jazyky. To znamená, že v kóde sa až tak nestaráme o to, ako zmeníme text na obrazovke, ale iba definujeme podmienky pre jeho vypísanie.
- Aplikácie v React Native sa píšu v jazyku JavaScript. Používajú sa natívne komponenty a ich API je unifikované pre všetky platformy. To znamená, že pre vývoj používame JavaScriptové komponenty, ktoré obaľujú (wrapper) natívne komponenty.
- React Native používa pre štýly CSS, alebo SCSS, čo znamená že môžme elementy štylizovať veľmi príjemnou a rozšírenou technológiou. Napr. narozdiel od Androidu, kde sa štylizovanie robí v jazyku XML a je špecifické iba pre Android.
- Vývoj v React Native je nesmierne rýchly aj vďaka funkcii *HotLoading*. Počas

vývoja nie je potreba aplikáciu zdĺhavo prekompilovávať, ale po zmene kódu sa zmeny automaticky hotloadujú do zariadenia na ktorom vyvýjame. Zmenu v UI alebo logike kódu tak môžeme testovať prakticky okamžite počas vývoja.

- React Native je open-source a teší sa veľkej obľube v komunite, takže vývojari preňho vytvárajú ďalšie rozširujúce knižnice. Výhodou frameworku je že umožňuje použitie akejkoľvek JavaScript knižnice.

Nasledujúci krátky príklad ukazuje jednoduchú aplikáciu ktorá ukazuje blikajúci text. Na prvom riadku je vidieť importovanie základného prvku Component z React knižnice a následne importovanie natívnych komponent ako Text a View. Následne si vytvoríme jednoduchú komponentu Blink. V jej konštruktore je vidieť premennú *props*, ktorá znamená property, ktorou môžeme parametrizovať komponentu pri jej používaní, to je práve text pridaný na riadku 29 a 30. V konštuktore je vidieť aj *state*. Každý Component môže definovať *state*, ten by mal definovať vnútorný stav komponenty. Používanie parametru *state* zaručí, že pri jeho zmene sa React Native postará o efektívne prekreslenie zmeny do UI. V tomto príklade definujeme interval pre výpis, v reálnej aplikácii by sme menili stav po prijatí dát zo serveru. Neskôr za konštuktorom definujeme metódu render(). Túto metódu musí obsahovať každá UI komponenta, na základe tejto metódy sa vykresluje komponenta do UI. Na riadku 25 už registrujeme jednoduchú aplikáciu, kde používame predtým vytvorenú komponentu Blink. Registrovanie aplikácie je obvykle v kóde iba na jednom a to spravidla počiatočnom mieste, kde sa spúšťa aplikácia.

```
1  import React, { Component } from 'react';
2  import { AppRegistry, Text, View } from 'react-native';
3
4  // Create reusable component
5  class Blink extends Component {
6    constructor(props) {
7      super(props);
8      this.state = {showText: true};
9
10     // Toggle the state every second
11     setInterval(() => {
12       this.setState(previousState => {
13         return { showText: !previousState.showText };
14       });
15     }, 1000);
16   }
17
18   // Define render method
19   render() {
20     // If showText in state is true show text from props.text
```

```
21      let display = this.state.showText ? this.props.text : ' ';
22      return (
23        <Text>{display}</Text>
24      );
25    }
26  }
27
28
29  export default class BlinkApp extends Component {
30    render() {
31      return (
32        <View>
33          <Blink text='I love to blink' />
34          <Blink text='Yes blinking is so great' />
35        </View>
36      );
37    }
38  }
39
40  // Register application
41  AppRegistry.registerComponent('AwesomeProject', () => BlinkApp);
```

Výpis 5.1: Príklad jednoduchej aplikácie v React Native [16]

Framework React Native ponúka jednoduché rozhranie, vysoký výkon natívnych komponentov a navyše práca s ním je veľmi príjemná. Pre riešenie našej aplikácie by sme sa pre klientskú časť priklonili práve k použitiu React Native technológie, aj kvôli vyššie spomenutým pozitívam.

5.1.4 Redux stavový kontajnér

Redux je malá open-source JavaScript knižnica, ktorú je možné použiť napríklad s Reactom. Keďže je React zameraný na UI a jeho vývoj, používa sa na uchovávanie stavu aplikácie. Redux je predikovateľný stavový kontajnér pre JavaScriptové aplikácie. Pomáha písať aplikácie, ktoré sa správajú konzistentne, bežia v rôznych prostrediach a sú jednoduché na testovanie. Redux používa *akcie* - *actions*, *reduktory* - *reducers* a *úložisko* - *store*.

Jednoducho vysvetlené, Redux ako *stavový kontajnér* - *state container* uchováva všetky dáta aplikácie na jednom mieste, prístupné odkiaľkoľvek z aplikácie. Stav úložiska je možné iba čítať, jediná cesta ako zmeniť stav, je odoslať akciu do reduktora. Stav môže byť zmenený potom iba reduktormi, ktoré sú čisté JavaScript funkcie. Reduktory Reduxu vezmú predchádzajúci stav úložiska a prijatú akciu a vytvoria nový stav pre úložisko. Akciami a reduktormi sa tak vytvorí sada možných

operácií ktoré môžme vykonávať nad úložiskom a to ho robí skutočne silným nástrojom, pretože v každom okamihu môžme predikovať nasledujúci stav, odvýjajúci sa od toho aktuálneho.

Z počiatku sa môže zdať Redux náročný, no opak je pravdou. Je to veľmi jednoduchá knižnica a najzložitejšie je asi pochopiť úlohu akcií, reduktorov a úložiska. Počiatočná zložitosť pravedpodobne vychádza z nevhodne zvolených názvov týchto častí, ktoré zvolili jej vývojari Dan Abramov a Andrew Clark. Dnes je však najpoužívanejšou knižnicou pre uchovávanie stavu v tejto oblasti.[17] O Reduxe sa hovorí, že nie je skvelý pre rýchle vytváranie jednoduchých aplikácií, ale je skvelý pretože robí zložité aplikácie jednoduchými.

Akcie

Akcia je čistý JavaScript objekt, ktorý v sebe nesie dáta, ktoré majú byť uložené do úložiska. Sú jediným zdrojom informácií pre úložisko aplikácie, iný spôsob zmeniť úložisko nie je (iný spôsob existuje, ale je proti pravidlám používania Reduxu). Akcie musia mať property *type*, ktorá indikuje typ akcie, ktorá sa vykoná na úložisku. Typy sú zvyčajne definované ako *string* konštanty a sú zdieľané s reduktormi.

```
1 const ADD_TODO = 'ADD_TODO'
2
3 {
4   type: ADD_TODO,
5   payload: 'Build my first Redux app'
6 }
```

Výpis 5.2: Akcie Reduxu[17]

K akciám ešte existujú *Tvorcovia akcií* - *Actions creators*, ktoré robia presne to ako sú pomenované, sú to obyčajné JavaScriptové funkcie ktoré vytvárajú akcie. V praxi sa hojne používajú, pretože vytváranie akcií zakaždým samostatne, by bolo veľmi neprehľadné a neudržateľné.

```
1 function addTodo(text) {
2   return {
3     type: ADD_TODO,
4     payload: text
5   }
6 }
```

Výpis 5.3: Tvorba akcií Reduxu[17]

Reduktory

Reduktor je čistá JavaScriptová funkcia. Reduktory špecifikujú, ako sa zmení stav úložiska aplikácie v odozve na akciu odoslanú do úložiska.

```
function todoApp(state = initialState, action) {
  switch (action.type) {
    case ADD_TODO:
      return Object.assign({}, state, {
        todos: [
          ...state.todos,
          {
            text: action.payload,
            completed: false
          }
        ]
      })
    default:
      return state
  }
}
```

Výpis 5.4: Reduktory Reduxu[17]

Úložisko

V predchádzajúcich častiach boli popísané akcie ako objekty, ktoré majú za úlohu svojim stavom popísať "čo sa stalo". Reduktory majú potom za úlohu zmeniť stav úložiska v závislosti na týchto akciách. Úložisko je znova objekt, ktorý dopĺňa skladačku akcií a reduktorov Reduxu. Má za úlohu držať stav aplikácie, je to opäť jednoduchý JavaScriptový objekt. Úložisko umožňuje prístup ku stavu pomocou funkcie *getState()* a umožňuje zmenu stavu pomocu akcií, ktoré sú vyvolané funkciou nad úložiskom *dispatch(akcia)*.

5.1.5 Realm databázový systém

Realm databáza je open-source NoSQL databázová knižnica optimalizovaná pre použitie v mobilných zariadeniach. Realm databáza nepoužíva klasické ORM ako sa môže zdať. Realm je tzv. objektová databáza, čo znamená že dáta sú reálne uložené v Realm ako objekty. Vďaka tomu je Realm údajne až 10-krát rýchlejší než konvenčné SQL riešenia.[15] Okrem toho má Realm niekoľko ďalších výhod:

- Ukladanie natívnych objektov. Realm databáza má podporu klasických jazykov pre vývoj mobilných aplikácií vrátane programovacích jazykov Swift, Java,

Objective-C, C#, a JavaScript(React Native). Objekty ktoré ukladáme v Realm sú vlastne tie objekty s ktorými pracujeme inde v kóde. Pracuje sa teda bez serializácie dát, vytvárania kópií, s dátami sa pracuje priamo. K dátam Realmu sa pristupuje "lazy" prístupom, to znamená, že dáta nie sú držané v pamäti RAM.

- Realm je multi-platformový. Pokiaľ neukladáme objekty v špecifické pre nejakú platformu, dáta môžu byť synchronizované naprieč operačnými systémami. Je tiež možné Realm dáta skopírovať medzi platformami a používať kdekoľvek.

Práca s Realm databázovou technológiou je skutočne jednoduchá. Najprv je potrebné nadefinovať modely v databáze pomocou schém. Veľkou výhodou je práve ako už bolo spomenuté používanie natívnych typov objektov. To znamená, že ak píšeme kód v React Native(JavaScript), Realm databáza dokáže pracovať priamo s týmito objektami a nám odpadá parsovanie a ďalšie úskalia klasických SQL databázi. Pri klasických ORM riešeniach je zvyčajne potrebné implementovať nejaký objekt pre prístup k dátam(DAO), ktorý sa nám stará o správne napĺňanie štruktúr dátových typov daného programovacieho jazyka práve z SQL databázy a naopak. Realm je napísaný v C++ a celé jeho jadro je optimalizované takže svojým výkonom presahuje klasické SQL riešenia ako napr. SQLLite na mobilných platformách. Nižšie je krátka ukážka zdrojového kódu práce s Realm databázou.

```
1  const Realm = require('realm');
2
3  // Define your models and their properties
4  const CarSchema = {
5    name: 'Car',
6      properties: {
7      make:  'string',
8      model: 'string',
9      miles: {type: 'int', default: 0},
10   }
11 };
12
13 const PersonSchema = {
14   name: 'Person',
15   properties: {
16     name:     'string',
17     birthday: 'date',
18     cars:     'Car[]',
19     picture:  'data?' // optional property
20   }
21 };
22
```

```
Realm.open({schema: [CarSchema, PersonSchema]})
  .then(realm => {
    // Create Realm objects and write to local storage
    realm.write(() => {
      const myCar = realm.create('Car', {
        make: 'Honda',
        model: 'Civic',
        miles: 1000,
      });
      myCar.miles += 20; // Update a property value
    });

    // Query Realm for all cars with a high mileage
    const cars = realm.objects('Car').filtered('miles > 1000');

    // Will return a Results object with our 1 car
    cars.length // => 1

    // Add another car
    realm.write(() => {
      const myCar = realm.create('Car', {
        make: 'Ford',
        model: 'Focus',
        miles: 2000,
      });
    });

    // Query results are updated in realtime
    cars.length // => 2
  })
  .catch(error => {
    console.log(error);
  });
```

Výpis 5.5: Príklad použitia Realm databázy v JavaScript [15]

5.1.6 PouchDB open-source framework

PouchDB je open-source JavaScript databáza inšpirovaná Apache CouchDB[19], ktorá bola vytvorená aby pomohla vývojárom stavať aplikácie ktoré budú pracovať rovnako offline ako aj online.[18] PouchDB je v podstate JavaScriptová implementácia CouchDB, keďže je CouchDB niečo ako samostatný softvér a implementovať ho priamo do aplikácie je náročné. Špecifiká CouchDB budú detailnejšie popísané neskôr v časti 5.2.3. Keďže PouchDB kopíruje CouchDB, všetky mechanizmy sú teda

rovnaké a databázy sú plne komatibilné. Nasledujúca špecifikácia platí teda pre obe databázy, PouchDB ako aj CouchDB.

Je otázka aký má zmysel používať inú databázu pri všetkých možnostiach ako MongoDB, PostgreSQL, MySQL a ďalších mnohých. PouchDB ponúka možnosť komunikovať s databázou priamo s klientom. Komunikácia prebieha cez HTTP. API spĺňa štandardy RESTu, je teda s databázou možné komunikovať aj napríklad z webového prehliadača, alebo akýmkoľvek REST klientom ako napr. Postman. Najväčšou výhodou databázy je ale synchronizácia, je navrhnutá od počiatku tak, aby bola databáza optimalizovaná pre synchronizáciu. PouchDB používa ale odlišnú konvenciu na rozdiel od SQL. Nasledujúca tabuľka ukazuje používanú konvenciu v databáze:

Tab. 5.1: Koncepcia PouchDB databázy [18]

SQL koncept	**PouchDB koncept**
tabuľka	*nie je ekvivalent*
riadok	dokument (document)
stĺpec	časť dokumentu (field)
primárny kľúč	primárny kľúč (_id)
index	pohľad (view)

Dokument

PouchDB je NoSQL databáza čo znamená, že môžme ukladať neštrukturované dokumenty namiesto explicitne špecifikovanej schémy tabuľky. Dokument teda môže byť akýkoľvek, no pre JavaScriptovú aplikáciu môže vyzerať napríklad takto:

```
1  var doc = {
2    "_id": "mittens",
3    "name": "Mittens",
4    "age": 3,
5    "hobbies": [
6      "chasing laser pointers",
7      "lookin' hella cute"
8    ]
9  };
10 db.put(doc);  // store document to DB
11
12 db.get('mittens').then(function (doc) { // get the document
13   console.log(doc);
14 });
```

Výpis 5.6: Príklad dokumentu PouchDB [18]

Revízia dokumentu

Dokument po uložení do databázy dostáva unikátny identifikátor revízie *_rev*. Je to náhodne generovaný identifikátor, ktorý sa mení kedykoľvek, keď je dokument vytvorený, alebo zmenený. Po vytiahnutí uloženého dokumentu z databázy teda dostaneme dokument, ktorý už obsahuje aj jeho revíziu ako je vidieť nižšie. Revízie dokumentov sa následne tiež používajú pri synchronizácií databázy.

```
{
  "name": "Mittens",
  "age": 3,
  "hobbies": [
    "chasing laser pointers",
    "lookin' hella cute"
  ],
  "_id": "mittens",
  "_rev": "1-bea5fa18e06522d12026f4aee6b15ee4"
}
```

Výpis 5.7: Revízia dokumentu PouchDB [18]

Replikácia

PouchDB a CouchDB boli navhrnuté pre jediný hlavný zmysel a to je synchronizácia. Začiatky s CouchDB môžu byť náročné, pretože nepracuje ako žiadná iná databáza, preto je zložitejšie sa v nej zorientovať. CouchDB ale bola navrhnutá s ohľadom na synchronizáciu a to je miesto kde exceluje. Používa sa pojem replikácia, teda všetky databázy sa udržiavajú v rovnakom stave - synchronizujú sa.

CouchDB nepoužíva master-slave architektúru, ale je multi-master, všetky databázy majú rovnaké práva. Synchronizácia naprieč viacerými uzlami nie je vôbec jednoduchá záležitosť, už len kvôli rôznej rýchlosti Internetového pripojenia, keď dáta prídu v odlišnom čase na rôzne uzly čo vytvára priestor na konflikty. V prípade konfliktu, CouchDB zvolí samostatne výhercu na základe revízie. Konflikty ale ostávajú uložené v strome revízií(podobne ako u stromu histórie Gitu), takže sa k nim dá vrátiť, alebo ich môžme proste ignorovať. PouchDB umožňuje tiež implementovať funkciu, ktorá sa postará o vyriešenie konfliktu, takže výsledok nebude iba v rukách frameworku.

Vzhľadom k CAP teorému (Consistency - Availability - Partition Tolerant), CouchDB je databáza, ktorá spĺňa dostupnosť a toleranicu rozdelenia teorému. To znamená že je rozdelená na časti, každý uzol je dostupný, a je iba eventuálne konzistentná. CAP teorém je bližšie popísaný v časti 5.2.

S PouchDB je vďaka jeho jednoduchému API práca veľmi jednoduchá, podobne je tomu v prípade synchronizácie. V našom prípade je zaujímavá obojsmerná synchronizácia, čo znamená, že v prípade vykonania zmeny na lokálnej databáze, aby sa tieto zmeny replikovali na vzdialenú databázu. Rovnako naopak v prípade zmeny na vzdialenej databáze chceme aby sa tieto zmeny replikovali do našej lokálnej databázy. Takisto v prípade nášho systému chceme, aby sa dáta na klientských aplikáciach menili reálne v čase, to nám PouchDB v kombinácii s CouchDB vie ponúknuť.

```
var localDB = new PouchDB('mylocaldb')
var remoteDB = new PouchDB('http://localhost:5984/myremotedb')

// we can set direction of replication
localDB.replicate.to(remoteDB);
localDB.replicate.from(remoteDB);

// or even bidirectional live replication
localDB.sync(remoteDB, {
  live: true,
  retry: true
}).on('change', function (change) {
  // something changed
}).on('paused', function (info) {
  // replication was paused, usually because of a lost connection
}).on('active', function (info) {
  // replication was resumed
}).on('error', function (err) {
  // totally unhandled error (shouldn't happen)
});
```

Výpis 5.8: Replikácia databázy PouchDB [18]

5.2 Serverová časť

Hlavnou úlohou serverovej časti pre naše riešenie je synchronizovať dáta medzi databázami klientov. Je potreba zvážiť, či je vhodné vydať sa cestou vlastného návrhu a vlastného riešenia, čo dáva maximálnu možnú kontrolu nad riešením. Alebo je druhá možnosť použiť nejaké hotové a overené riešenie, nie je totiž dôležité snažiť sa za každú cenu vymyslieť niečo čo je už overené a funguje.

CAP teorém

CAP teorém (Consistency - Availability - Partition Tolerant) je aplikovateľný pre všetky sieťovo distribuované systémy ktoré udržiavajú nejaký stav, a dnes je uzná-

vaným nástrojom pre návrhárov takýchto distribuovaných systémov. Tvorca CAP teoremu Eric Brewer, tvrdí že každý systém ktorý zdieľa dáta cez sieť je kompromisom medzi konzistenciou, dostupnosťou a toleranciou rozdelenia.[21] V roku 2002 výskumníci z Massachusettskej technickej univerzity tento teorém dokázali.[20] Teorém tvrdí, že každý systém, ktorý zdieľa dáta po sieti, môže garantovať iba dve z nasledujúcich troch vlastností:

- **Konzistencia** - garancia, že každý uzol v distribuovanom systéme má ten samý, najposlednejší úspešný zápis. Konzistencia teda popisuje to, že každý uzol má rovnaké dáta.
- **Dostupnosť** - Každý neporušený uzol vráti odpoveď na čítanie a zápis v primeranom čase. Aby bol systém dostupný, musí každý jeho uzol odpovedať v primeranom čase.
- **Tolerancia rozdelenia** - Systém bude fungovať naďalej a udržiavať jeho konzistenciu, v zmysle jeho častí na sieti. Znamená to, že distribuované systémy ktoré garantujú toleranciu rozdelenia môžu byť bez problémov obnovené v prípade opravy, napríklad z iného uzla. V jednoduchosti teda pravidlo hovorí o tom, že dáta môžu byť distribuované na viacerých uzloch či už sa jedná o servery, alebo klientov.

Každý distribuovaný systém, ktorý zdieľa dáta po sieti musí mať toleranciu rozdelenia, to vychádza už z podstaty sieťového pripojenia. Časti siete a zahodené sieťové správy sú súčasťou reálneho sveta a musia byť patrične ošetrené. Z toho vyplýva, že návrhári takéhoto sieťovo distribuovaného systému si musia vybrať medzi dodržaním konzistencie, alebo dodržaním dostupnosti. Nie je možné dodržať všetky tri časti teorému súčasne. Pri návrhu sa teda najčastejšie počíta s voľbou, kedy má systém toleranciu rozdelenia, každý uzol je dostupný, a je iba eventuálne konzistentný.

5.2.1 Node.js serverová aplikácia

Javí sa ako vhodná alternatíva napísať si vlastné serverové riešenie. Byť závislý na riešení tretích strán môže priniesť veľké komplikácie a tiež nám to nedáva dostatočnú kontrolu nad riešením.

Pre sieťovú komunikáciu medzi klientmi a serverom by bolo vhodné používať technológiu WebSocket. To nám umožní veľmi efektívne zdieľať dáta medzi pripojenými zariadeniami v reálnom čase. V prípade, že by sme nepoužili WebSocket, ale napríklad klasické REST API, znamenalo by to, že klienti by sa museli dotazovať na nové zmeny napríklad každú minútu, to by so sebou prinieslo viacero negatív:

- Zvýšila by sa záťaž na server, kvôli potrebe autentifikovať každý dotaz.
- Zbytočne by sa zvýšil objem prenášaných dát práve kvôli potrebe prenášať autentifikačné meta data v každej správe.

- Užívatelia by nemali pocit že aplikácia sa mení v reálnom čase. Ak by jeden čašník urobil objednávku, druhému by sa táto zmena mohla zobraziť až po minúte. Pre našu aplikáciu sa hodí UX aký má užívateľ pri používaní chatu, kde sa zmeny prejavujú medzi zariadeniami používateľov okamžite, v reálnom čase.

Kvôli vyššie spomenutému je teda vhodné použiť práve technológiu WebSocket.

Pre prenos dát medzi klientom a serverom by bolo vhodné použiť Json formát dát. Ten je momentálne najširšie používaný formát kvôli jeho skvelej čitateľnosti a jednoduchosti. Napríklad oproti zápisu XML má viacero výhod, ako je pomer prenesených dát vs. potrebná gramatika zápisu. To znamená, že ak niekedy potrebujeme poslať krátku správu, pri zápise XML je potreba posielať zbytočne dlhé identifikátory a popisky, a posielanie je tak neefektívne.

Z vyššie popísaných špecifík vyplýva, že by bolo najvhodnejšie zvoliť pre vývoj backendu jazyk JavaScript. To vyplýva z toho, že budeme prenášať dáta v Json, a takisto aj klientské aplikácie budú používať JavaScript(React Native) ako hlavnú technológiu. Touto voľbou sa vyhneme potrebe nadbytočného parsovania. Pre vývoj backendu by bolo vhodné teda použiť technológiu Node.js.

Ako databázový systém pre backend by bolo vhodné zvoliť NoSQL databázu, s podporou ukladania Json objektov. Keďže sa jednotlivé položky databázy môžu vytvárať na klientoch, potom po synchronizácii preniesť na server a nakoniec na ďalších klientov, používame ako primárny kľúč UUID a nie je vhodné používať v tomto prípade inkrementovaný primárny kľúč. Tu sa teda ponúka riešiť to databázou Realm alebo MongoDB. Práve druhá spomenutá má dobrú podporu v Node.js a je to pomerne aj štandard medzi cloudovými poskytovateľmi, že ponúkajú prostredie kombinujúce Node.js - MongoDB za výhodnejšiu cenu.

5.2.2 Realm Object Server

Firma Realm ponúka riešenie kompletnej Realm platformy, ktorá obnáša už spomínanú open-source Realm databázu a spoplatnené serverové riešenie Realm Object Server.[15] Realm je veľmi obľúbený obzvlášť v oblasti tvorby mobilných aplikácií, práve kvôli tomu, že ponúka veľmi výkonnú ľahkú open-source databázu. V mnohých prípadoch, potom pri vývoji aplikácií, v snahe ušetriť čas, je použité aj ich serverové riešenie. Realm Object Server je riešenie, ktoré značne uľahčuje prácu, tým že nie je potreba implementovať registráciu užívateľov, ich autentifikáciu ale ani synchronizáciu dát. To sú časti, ktoré sú pre mnohé aplikácie dookola implementované a Realm ich práve ponúka.

Realm platforma ponúka neplatenú verziu Developer a rozšírenú platenú verziu Enterprise. V čase písania práce sa ale licenčné podmienky zmenili, čo práve uka-

zuje nevýhodu použitia licencovaných riešení tretích strán. Po komunikácii s firmou Realm, vyplynulo, že do ich serverového riešenia implementovali od verzie 2.6.3 overovanie tokenov, ktoré zaručí plnú kontrolu firmy nad serverom. Znamená to teda, že aj keď je Realm Object Server možné nasadiť na vlastnom serveri, stále budú mať vďaka tokenu kontrolu nad jeho prevádzkovaním oni a v prípade zmien podmienok alebo neuhradenia paušálneho poplatku, server prestane fungovať.

Myslíme si, že nie je vhodné použiť žiadnu licencovanú platformu tretích strán ako Realm (alebo aj resp. Firebase od firmy Google), práve z dôvodu že sa podmienky používania môžu kedykoľvek zmeniť, a takisto je tam potreba neustále platiť paušálne poplatky. Práve z toho dôvodu je vhodnejšie implementovať vlastné riešenie, ktoré nám dá plnú kontrolu nad riešením, alebo použiť nejaké iné open-source serverové riešenie.

5.2.3 CouchDB open-source framework

CouchDB je NoSQL databáza vytvorená v roku 2005 Damienom Katzom, momentálne je v správe Apache Software Foundation. CouchDB je open-source databáza, ktorú je možné prevádzkovať na vlastnom serveri a nie je nijako naviazaná na žiadnu firmu za účelom biznisu. CouchDB nie je možné implementovať priamo do klientských aplikácií, keďže to nie je knižnica prispôsobená na rôzne platformy. Databáza definuje otvorený Couch replikačný protokol, ktorý je možné implementovať a tým vytvoriť plne kompatibilnú databázu, tento protokol napríklad už implementuje PouchDB databáza. Couch replikačný protokol umožňuje bezproblémový prenos dát medzi serverom a mobilnými telefónmi, alebo webovými prehliadačmi čo umožňuje vytvárať aplikácie so žiadúcou offline-first UX. CouchDB obsahuje všetky mechanizmy popísané pre PouchDB v časti 5.1.6. Aj keď boli jednotlivé mechanizmy popísané už pri PouchDB, práve CouchDB je zdrojom týchto skvelých mechanizmov, a je to skutočne dobre navrhnutý a fungujúci databázový systém.[19]

6 Návrh architektúry systému

Je veľmi dôležité zvážiť všetky predchádzajúce zistenia a na ich základe navrhnúť architektúru riešenia. Nesprávne zvolená architektúra dátového modelu aplikácie by mohla mať vážne dopady na jej neskorší rozvoj. Pre ukladanie dát bude použitá databáza a je potrebné riešiť aký typ databázy používať.

Môžme zvoliť prístup temporálnej databázy, ktorá vlastne zachytáva vývoj a zmeny dát uložených v databáze. Kvôli zložitosti a potrebe synchronizovať medzi zariadeniami by sme ale tento prístup nevolili. Takisto, neuvažujeme, že budeme zmeny revertovať, preto nemá až taký význam uchovávať všetky zmeny ktoré sme vykonali.

Druhým možným prístupom je, aby každý záznam v databáze obsahoval čas poslednej zmeny *modifiedAt*, vďaka tejto vlastnosti vieme jednoducho synchronizovať a aj meniť dáta v databázi. Klient si bude uchovávať čas poslednej synchronizácie a po žiadaní o nové dáta s parametrom *If-modified-since* by dostal dáta ktoré sú pre neho relevantné. Takisto naopak, keď by odovzdával dáta na server, s lokálnymi zmenami vykonanými po poslednej synchronizácii, my na serveri zohľadníme tieto zmeny len do dát, ktoré neboli medzičasom zmenené. Týmto spôsobom sa jednoducho vyhneme možným konfliktom a mergovaniu dát alebo potrebe ďalšej logiky pri synchronizácií klientských lokálnych zmien. Praktika používania parametru *modifiedAt* pre dáta, je široko využívaná a používa ju napríklad aj Realm.

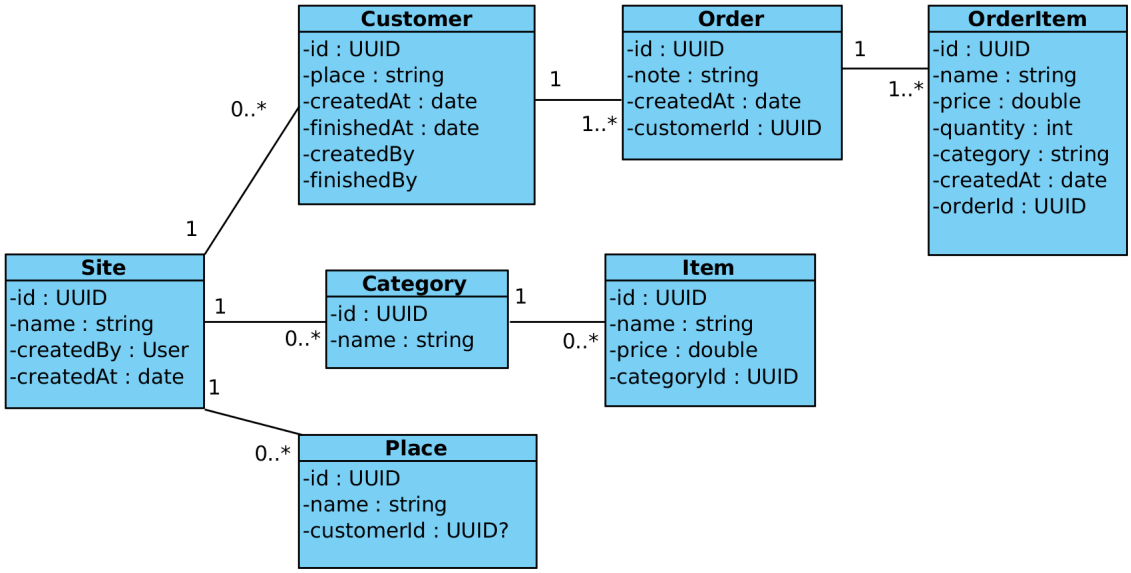
Z požiadávok na systém vyplýva, že na každom klientovi je potrebné uchovávať všetky dáta pre daného používateľa. Dáta budú replikované zo servera. Tento prístup chceme zvoliť práve kvôli možnosti offline prezerania štatistík. Zakisto nie všetci zákazníci musia využívať aj online režim a môžu fungovať čisto iba ako samostatný predajca v offline režime a pritom mať možnosť využívať všetky funkcie aplikácie.

V počiatočnej fáze vývoja synchronizácie sme začali implementovať vlastné riešenie synchronizácie. Z dôvodu, že v čase vývoja bola používaná lokálna databáza Realm, ktorá uchováva dáta ako objekty, to vytvorilo obrovskú zložitosť v syncrhonizácií dát. Problém spočíval v tom, že objekty boli vnorené, viacúrovňové a obsahovali teda objekty ďalších typov. Synchronizácia cez viacero úrovní bola veľmi náročná, preto sme od nej upustili a priklonili sa k použitiu SQLite databázy v kombinácii s PouchDB a CouchDB riešením, ktoré bude aj primárne popísane v tejto časti návhru architektúry. Do budúcna by bolo vhodné zvážiť znovu použitie Realm databázy, kvôli jej rýchlosti a rozhodne vyššej stabilite, no nepoužívať referencované vnorené objekty, ale naopak prístup podobný ako sa používa vo svete relačných databáz a to formou cudzých kľúčov.

6.1 Klientská časť systému

Dátový model

Architektúra databázy klientskej časti bude veľmi podobná tej serverovej. Obidve strany budú uchovávať viacmenej rovnaké dáta. Na obrázku 8.1 nižšie, je uvedený návrh architektúry dátového modelu klientskej časti systému.



Obr. 6.1: Dátový model klientskej časti

Počítame s tým, že užívateľ si vytvorí v aplikácii pobočku. Pobočka môže potom obsahovať miesta, a položky k predaju. Užívateľ vytvára objednávky položiek na jednotlivých miestach. Pre tento bežný scenár slúžia dátové modely *Category* - *Kategória*, *Item* - *Položka* a *Place* - *Miesto* ako je vidieť na návrhu. Miesto udržiava záznam o tom, či je na ňom aktuálne zákazník.

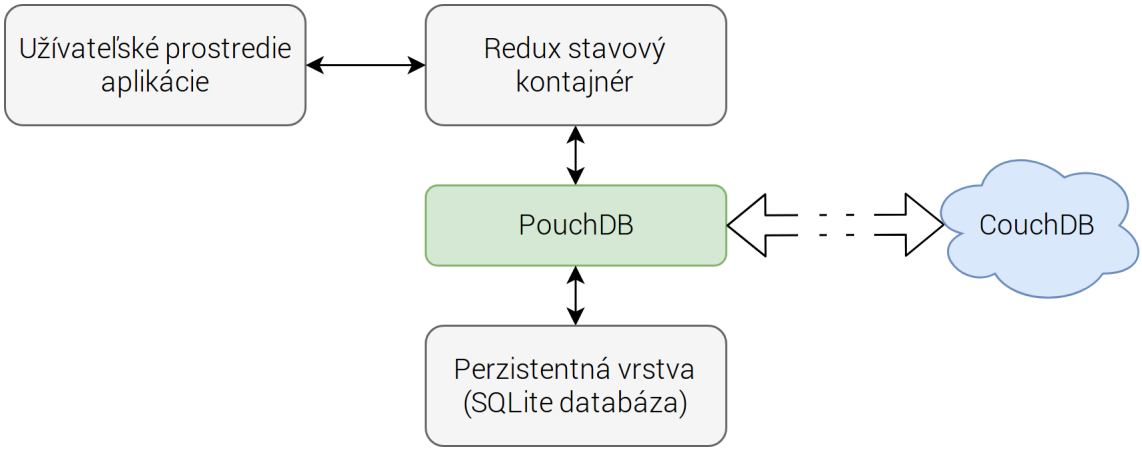
Ďalšou časťou je scenár vytvárania objednávky. Keď obsluha prevádzky urobí objednávku na prázdnom mieste, je automaticky vytvorený nový zákazník s jednou objednávkou. Zákazník ale môže mať viacero objednávok, ktoré má obsluha stále možnosť pridávať, až kým zákazník nezaplatí. Každá objednávka môže obsahovať viacero položiek. Pre tento scenár boli vytvorené dátové modely *Customer* - *Zákazník*, *Order* - *Objednávka* a *Položka objednávky* - *OrderItem*. V tejto časti sa teda snažíme zaznamenávať všetku činnosť spojenú s predajom. Môžeme si všimnúť že dáta niesú referencované. Napríklad model zákazníka uklada miesto ako string. To robíme z toho dôvodu aby sme zachovali stav aký bol v čase vytvárania objednávky. Keby sme uchovávali referencie, dáta pre štatistiky by sa nám mohli časom meniť napríklad tým že užívatelia zmenia názov miesta. Bolo by tiež naivné spoliehať sa,

že dáta v referenciách sa nemôžu stratiť alebo poškodiť. Z toho dôvodu ukladáme tieto dáta o objednaných položkách znova.

Architektúra ešte musí počitať s uchovávaním užívateľov a pobočkiek, ktoré môžu používatelia v rámci aplikácie vytvárať, táto časť nie sú súčasťou tejto práce.

Tok dát v systéme

Aplikácia bude napísaná v React Native frameworku. Keďže je React Native framework viac zameraný na *užívateľské prostredie – User Interface* (UI), bude použitý Redux pre uchovávanie dát - stavu aplikácie. Použitie Reduxu stavového kontajnéra sprehľadní aplikáciu a biznis logika by mala byť kompletne smerovaná do neho. Úložisko Reduxu bude držať objekty *Category*, *Item*, *Place*, *Customer*, *Order* a *OrderItem*. Ako bolo uvedené v tabuľke 5.1, súvislosti medzi SQL konceptom a CouchDB konceptom. Pre našu aplikáciu si zvolíme to, že každý dátový typ bude mať svoj dokument. Každý typ objektu teda bude predstavovať dokument PouchDB, a z Reduxu sa bude stav reflektovať tam a späť do PouchDB. PouchDB sa postará o replikáciu dát na cloud. Nakoniec, aby stav aplikácie ostal lokálne zachovaný aj po jej ukončení, bude použitá ako perzistentná vrstva SQLite databáza.



Obr. 6.2: Architektúra aplikácie pre podporu predaja

6.2 Serverová časť systému

Serverová časť systému by mala mať veľmi podobnú architektúru ako klientská časť aplikácie. V tomto štádiu a pre splnenie zadania tejto práce má serverová časť skutočne slúžiť len ako prostredník pre výmenu dát medzi klientskými aplikáciami. Preto bude dátový model serverovej časti rovnaký s tými klientskými. Pre synchronizáciu dát bude použitý CouchDB framework, ktorý sa bude starať o replikáciu

dát. Implementácia serverovej časti bude teda spočívať hlavne v správnom nakonfigurovaní CouchDB.

Očakávame, že neskôr bude k serverovej časti pridaná funkcia registrácie a autentifikácie užívateľov a serverová časť bude musieť byť schopná takisto držať dáta o viacerých pobočkách naraz a zároveň dáta synchronizovať len špecifickým klientom.

7 Implementácia klientskej časti systému

Ako už bolo spomenuté v kapitole 6 pred návhrom architektúry sme sa snažili overiť dostupné riešenia synchronizácie dát. Hlavnou myšlienkou bolo nevytvárať už overené koncepty, ale ušetriť tak čas a zdroje pri vývoji.

Nejaký čas sme venovali open-source projektu ParsePlatform,[26] ktorý bol skôr vyvýjaný firmou Facebook. Bohužiaľ sa ukázalo že je pre JavaScript platformu nepoužiteľný a obsahuje mnoho chýb. Celkovo bol problém docieliť čo i len jednosmernú synchronizáciu z klientskej aplikácie na server. Po niekoľko dňovom testovaní sme od tohto riešenia upustili.

Takisto bola otestovaná implementácia platformy Realm a ich cloudové riešenie Realm Object Server, no už pri vývoji sa ukázali problémy s použitím riešení tretích strán, keď firma zmenila spôsob spoplatnenia platformy. Preto sme od použitia Realm Object Server upustili.

Podstatne dlhšiu dobu sme sa venovali vývoju vlastného riešenia pre synchronizáciu, no to sa stalo príliž zložitým z dôvodu architektúry Realm databázy, ale tiež nášho návrhu dátového modelu. Problémy boli bližšie popísané v úvode kapitoly 6. Serverovú časť sme si napísali vo frameworku Node.js s použitím Realm databázy, takže sme mali rovnakú databázu ako na klientskej, tak na serverovej časti. Implementácia prebiehala podľa návrhu. Bol vytvorený WebSocket server a v reálnom čase si medzi sebou zariadenia vymieňali dáta. Problém ale nastal pri replikácii zložitých vnorených objektov. Bolo veľmi náročné vyhodnocovať zmeny vo vnorených objektoch a optimalizovať ich prenos po sieti. Hoci sme od riešenia upustili, z dôvodu časového deficitu a náročnosti, myslíme si že je toto riešenie najvhodnejšie, keďže nám dávalo obrovskú kontrolu nad riešením. Určite bude vhodné do budúcna použiť práve toto riešenie, za použitia inej architektúry databázy ako už bolo načrtnuté v kapitole..

7.1 Implementácia užívateľského rozhrania

Pred samotným vývojom prebiehal návrh obrazoviek na papier, tak aby aplikácia spĺňala dané požiadávky. Po tom by mal nasledovať návrh obrazoviek grafikom, čo sme my fiktívne urobili v kapitole 4.2. Ako vývojári, by sme pred vývojom mali premýšlať nad architektúrou aplikácie. Aktuálne rozšírená MVP(model-view-presenter) architektúra, ktorá je derivátom MVC(model-view-controler) sa zvykne používať naprieč mobilnými aplikáciami. To ale neznamená že stále je tou najlepšou voľbou, my ju nepoužijeme, tiež z dôvodu juniórnej znalosti Reactu Native v počiatkoch písania aplikácie. Každopádne počas písania kódu sa budeme riadiť clean-code paradigmami[22] a budeme sa snažiť písať kód čitateľný a udržateľný.

Užívateľské rozhranie aplikácie bolo implementované bez väčších problémov. Pre počiatok sme použili namockované dáta, tak aby sa nám UI jednoduchšie stavalo a testovalo. React Native používa pre štýlovanie CSS, takže práca zo štýlovaním bola veľmi príjemná. Pre navigáciu medzi obrazovkami a prechody medzi nimi sme použili komponentu React Navigation.[23]

Ako možné vylepšenie by bolo implementovanie podpory pre fotky predajných položiek. Je potreba si uvedomiť, že je jednoduchšie a rýchlejšie rozoznať pre užívateľa miniatúry, než písaný text. Z toho dôvodu, by bolo vhodné do dialógu pre vytváranie predajných položiek pridať možnosť fotky a tú uložiť v malom formáte ako Base64 do databázy. Ukladaním malého obrázku do databázy by bola hneď vyriešená synchronizácia medzi zariadeniami, a nebolo by potrebné riešiť synchronizáciu assetov. Tiež, nie je možné očakávať, že užívateľ fotografiu zaručenie použije, z toho dôvodu by bolo vhodné generovať miniatúram farby na základe textu položky. Podľa nášho názoru, už len použitie farieb by uľahčilo používanie oproti písanému textu. Nižšie je uvedený príklad generovania hexadecimálneho kódu farby na základe hashu textu.

```
function stringToColor(str) {
  var hash = 0;
  for (var i = 0; i < str.length; i++) {
    hash = str.charCodeAt(i) + ((hash << 5) - hash);
  }
  var color = '#';
  for (var i = 0; i < 3; i++) {
    var value = (hash >> (i * 8)) & 0xFF;
    color += ('00' + value.toString(16)).substr(-2);
  }
  return color;
}
```

Výpis 7.1: Príklad vytvorenia farby na základe textu

7.2 Implementácia Redux stavového kontajnéra

Pre uchovávanie stavu aplikácie sme použili Redux stavový kontajnér. Ten nám pomáha oddeliť biznis logiku od UI a udržiavať dáta na jednom mieste prístupné z rôzných miest aplikácie. Redux je predikovateľný stavový kontajnér, čo znamená že máme dané dostupné akcie, ktorými môže byť menený stav aplikácie, okrem nich by sa stav aplikácie nemal nijako inak nepredikovateľne meniť. Aplikáciu sme navrhli tak, že máme vlastné reduktory pre každý typ dát. Prakticky to znamená, že v aplikácii máme reduktory pre *Place*, *Category*, *Item*, *Customer* a *Order*. Pre tieto

reduktory máme spravených tvorcov akcií, ktoré sú po vytvorení vložené do daného reduktora. Reduktor následne na základe akcie ktorú prijal, zmení stav aplikácie. Nižšie je uvedený príklad implementácie vytvárania akcie pre vytvorenie nového miesta a takisto jeho reduktor, ktorý sa postará o zmenu stavu.

```
1  const CREATE_PLACE = 'create_place';
2  // define action creator
3  const createPlace = (name) => {
4    return (dispatch) => {
5      const place = {
6        id: uuid(),
7        name,
8        customerId: null
9      };
10     dispatch({ type: CREATE_PLACE, payload: place });
11     return Promise.resolve(place);
12   };
13 };
14 // define reducer
15 const initialState = {};
16 function placeReducer(state = initialState, action) => {
17   switch (action.type) {
18     case CREATE_PLACE:
19       return { ...state, [action.payload.id]: action.payload};
20     default:
21       return state;
22   }
23 };
```

Výpis 7.2: Príklad implementácie akcie a reduktora Reduxu

Pre vytváranie akcií sme použili knižnicu Redux Thunk.[24] Redux Thunk je zásuvný modul (middleware) pre Redux, ktorý nám umožňuje robiť vytváranie akcií, ktoré vracajú funkciu namiesto akcie. To má obrovskú výhodu pri tvorbe náročnejších operácií, pretože tým zanášame asynchronicitu do riešenia. Reduktor tak dostane funkciu, ktorá môže byť vykonávaná ľubovoľne dlho a po jej dokončení reduktor zmení stav aplikácie. V príklade implementácie reduktora pre vytvorenie miesta je tiež vidieť jednu záujimavú skutočnosť a to, že stav aplikácie udržiavaný v Reduxe má byť "immutable". Znamená to, že nie je správne meniť nejakú vlastnosť objektu, ale pri zmene časti objektu, musí byť znova nový objekt s danou zmenou. Je to vidieť na riadku 19, kde zámerne používame tzv. "spread" operátor JavaScriptu.

Aby sa uchoval stav aplikácie aj po jej vypnutí implementovali sme PouchDB, ktoré uchováva stav Redux kontajnéra. PouchDB sama o sebe je veľmi ľahká knižnica, ktorá nemá perzistentnú vrstu, jej výber implementácia je na vývojarovi. My

sme použili SQLite, ktoré nijak priamo nepoužívame, iba si do neho PouchDB ukladá dáta. Architektúru PouchDB databázy sme zvolili tak, že každý reduktor má svoj PouchDB dokument, podľa SQL konceptov sa dokument pre nás správa ako tabuľka.

7.3 Implementácia WebView pre štatistiky

Požiadávky na zobrazovanie štatistík sú jasné a to, že musia byť zobraziteľné na akomkoľvek zariadení. To znamená, že ak aj budeme mať užívateľa aplikácie ktorý nebude využívať serverovú časť, ten musí mať štatistiky tiež dostupné. Štatistiky by mali fungovať offline, čím pomôžeme odľahčiť záťaž na server a prenesieme ju na zariadenie užívateľa. Štatitiky by mohli byť v budúcnosti implementované aj do webového rozhrania tak, že si ich bude môcť manažér pozrieť na počítači. Z toho dôvodu vytvoríme štatistiky ako webovú aplikáciu, ktorá bude dostupná offline vo webovom okne aplikácie, pričom bude zobrazovať lokálne dáta aplikácie.

Na postavenie webovej aplikácie sme použili echnológie HTML, CSS a JavaScriptový framework React. Keďže sme používali React Native na vývoj mobilnej aplikácie, použitie Reactu dávalo logiku, pretože prechod medzi nimi je veľmi jednoduchý. React nám podstatne zjednodušil prácu, nakoľko sme nemuseli ručne nastavovať objekty v DOM HTML ale tie boli veľmi efektívne renderované React algoritmom. Pri práci sme postupovali tak, že sme najprv overili jednotlivé koncepty webovej aplikácie ako je prenos lokálnych dát do webového okna. Následne sme použili namockované dáta, postavili algoritmus na vyhodnocovanie dát a spravili funkčné UI. Následne sme dokončili most na prenos dát medzi natívnou aplikáciou a webovým oknom a vpustili do aplikácie reálne dáta.

Dôležitou časťou obrazovky štatistík sú grafy. Tu sme hľadali JavaScript knižnicu, ktorá nám pomôže kresliť grafy. Medzi najznámejšie patrí veľmi vyspelá a pokročilá knižnica D3.js [27] alebo open-source knižnica Chart.js [28]. Kvôli jednoduchosti a otvorenému kódu Chart.js sme sa priklonili k jej použitiu. Práca s Chart.js bola pomerne jednoduchá, po naštudovaní dokumentácie bola rýchlo použiteľná a jej implementácia nebola nijak náročná. Chart.js obsahuje rôzne druhy grafov, pričom komponentu už len stačí JavaScriptovým objektom nakonfigurovať a nasypať do nej správne dáta pre zobrazenie.

Nemenej náročnejšou časťou bolo vyhodnocovanie dát. Tu sme zvolili pravdepodobne jediný možný prístup a to je, že dáta sme museli jeden objekt po druhom prejsť a robiť nad nimi rôzne aritmetické operácie. Všetky uložené dátové objekty v našej aplikácii sú navrhnuté tak, že obsahujú časy vytvorenia a zmien, a tieto dáta sme hojne využívali za pomoci knižnice Moment.js [29] pre prácu s časom. Pri tvorbe algoritmu sme dbali na to, že cez dáta prebehneme iba raz a počas tohto

cyklu sa vypočítajú všetky potrebné údaje pre štatistiky. Vďaka ohľadu na efektívnosť, výpočet trvá veľmi krátko(rádovo jednotky milisekúnd pri stovkách objektov) a užívateľ nie je rušený dlhým čakaním.

Zložitejšou časťou bolo vymyslieť spôsob akým sa budú dáta vymieňať medzi natívnou aplikáciou a aplikáciou vo webovom okne. Pri programovaní natívnej Android aplikácie, by bolo možné použiť JavaScript interface implementovaný v SDK, ktorý injectuje natívne volania do webového okna. To umožní prevolávanie natívnych metód z webového okna. U React Native je situácia zložitejšia zrejme aj kvôli vyspelosti frameworku. React Native umožňuje poslať správu do HTML DOM aplikácie webového okna. DOM natívne podporuje zachytávanie rôzných udalostí, ako zobrazenie stránky *pageshow*, alebo často využívaný callback *load* po načítaní stránky, a mnoho ďalších.[30] Udalosť ktorú sme použili, je udalosť *message*, informujúca o prijatí správy do DOM, ako ukazuje príklad implementácie event listenera pomocou JavaScriptu nižšie. Práve tak nám vznikolo premostenie a za použitia správ si vieme vymieňať dáta medzi natívnou časťou a webovým oknom aplikácie. Pri vývoji mostu sme sa inšpirovali príkladom implementácie na GitHube.[31]

```
1 window.document.addEventListener('message', function(e) {
2   console.log("message received from react native");
3 }
```

Výpis 7.3: Implementácia event listenera na udalosť ”message” v DOM HTML

8 Implementácia serverovej časti systému

Ako vyplynulo z analýzy, najlepšie by bolo prísť s vlastným riešením serverovej časti, aj keď za cenu väčšieho množstva práce. Zároveň ale, ak existuje dostatočne dobré open-source riešenie, nie je výhradou server implementovať vlastnými silami. V tejto časti bude popísané ako prebiehal vývoj vlastného riešenia serverovej časti systému. S akými problémami sme sa počas vývoja stretli a napokon k akému finálnemu riešeniu sme sa priklonili.

Ako už bolo spomenuté v úvode kapitoly 7, podobne ako pre klientskú časť, sme sa pokúšali implementovať zároveň aj serverovú časť. Vystriedali sme tak viacero riešení, z ktorých sa ako najlepšie zdala byť implementácia vlastného riešenia. Z dôvodu vysokej náročnosti, sme ale od implementácie vlastného riešenia upustili. Objavili sme veľmi jednoduchý a hlavne funkčný open-source projekt CouchDB od Apache Foundation, pre ktorý sme sa finálne rozhodli.

8.1 Implementácia vlastného riešenia

Pre implementáciu vlastného riešenia serverovej časti systému sme zvolili technológiu Node.js v spojení s databázou Realm. Vďaka tomu, že je Node.js postavené na JavaScripte, jednotlivé funkcie, ktoré sme napísali pre kontrolovanie dát mohli byť prepoužité takisto na klientskej aplikácii.

Architektúru databázy sme navrhli tak, že obsahovala stĺpce *modifiedAt* a *deletedAt*. Parameter *deletedAt* slúži len ako flag, že je daný záznam vymazaný. Celé riešenie stavalo na použití parametra *modifiedAt* tak, že na strane servera bol vytvorený jednoduchý algoritmus, ktorý porovnával či sa daný záznam nachádza v databáze, a ak áno, tak bol použitý čas v *modifiedAt* na základe ktorého sa použil novší záznam. Neskôr sme vytvorený algoritmus nasadili tiež aj na klientskú stranu aplikácie z jasných dôvodov. Nemôžme vedieť aký stav je na klientskej aplikácii a či obsahuje najposlednejšie dáta, preto sme akékoľvek zmeny broadcastovali na všetkých pripojených klientov a tak sa klientská aplikácia správala rovnako ako tá serverová, vyhodnocovala pre svoju databázu, ktoré dáta použije. Riešenie bolo multi-master. Ďalším scenárom je synchronizácia dát po pripojení klienta, ktorú sme sa rozhodli riešiť tak, že klient si udržiava čas poslednej synchronizácie *lastSync* a na základe toho ihneď po nadviazaní spojenia, sa dáta zosynchronizujú oboma smermi, znova prehľadávaním dát cez *modifiedAt*.

Z dôvodu, že chceme aby prebiehala synchronizácia dát v reálnom čase, zvolili sme spojenie pomocou technológie WebSocket. WebSocket nám umožňuje otvoriť spojenie medzi klientom a serverom a posielať medzi sebou správy, pričom na oboch stranách dostávame callback v prípade prijatej správy. WebSocket má aj nespornú

výhodu v tom, že ak klienta raz pri nadväzovaní spojenia overíme, nie je potreba autentifikovať každú správu a tým sa zníži množstvo nadbytočne prenesených dát. Nižšie je uvedený jednoduchý príklad WebSocket servera, na ktorom sme potom stavali ďalšiu logiku.

```
import express from 'express';
import http from 'http';
import WebSocket from 'ws';

const app = express();
// initialize simple http server
const server = http.createServer(app);
// initialize WebSocket server
const webSocketServer = new WebSocket.Server({ server });

webSocketServer.on('connection', (webSocket) => {
  webSocket.on('message', (message) => {
    // send received message back to client
    webSocket.send(`Hi, you sent -> ${message}`);
  });
  // send instantly feedback to incoming connection
  webSocket.send('Hi, you are connected to WebSocket server');
});

// start server
server.listen(process.env.PORT || 8999, () => {
  console.log(`Server started on port ${server.address().port}`);
});
```

Výpis 8.1: Jednoduchý WebSocket server v Node.js

Riešenie síce zneje dobre a aj čiastočne fungovalo, museli sme od neho upustiť z dôvodu veľkej náročnosti a ďalších už spomenutých dôvodov. Podľa nášho uváženia, by bolo toto riešenie najvhodnejšie, práve kvôli tomu, že máme plne pod kontrolou algoritmus synchronizácie a vieme pridávať ďalšie rôzne funkcie podľa vlastného uváženia. Tiež práca a nasadenie Node.js servera je veľmi príjemná záležitosť a je jednoduché nájsť poskytovateľov cloudových služieb s technológiou Node.js v palete. Do budúcna by bolo toto riešenie vhodné, hoci je potrebné ešte zvážiť rôzne prístupy z dôvodu že navrhnuté riešenie tiež môže obsahovať rôzne diery, ktoré môžu vyústiť do vzniku rozdielných stavov v dátach na jednotlivých uzloch.

8.2 Implementácia CouchDB

Nasadenie CouchDB je pomerne jednoduchá záležitosť.[19] Po inštalácii je databáza pripravená na používanie, už len s potrebou malých nastavení. Inštaláciu sme prevádzali na Linux OS. Na webových stránkach CouchDB kompletný návod inštalácie, v princípe je postup všetobecne známy. Po pridaní repository verzií CouchDB a ich verejného prístupového kľúča, môžeme CouchDB nainštalovať pomocou utility *apt*.

CouchDB defaultne vyčítava konfiguračné súbory z rôzných umiestnení v nasledovnom poradí:

1. *etc/default.ini*
2. *etc/default.d/*.ini*
3. *etc/local.ini*
4. *etc/local.d/*.ini*

Z dôvodu, že *default.ini* môže byť prepísané počas updatovania, alebo reinštalácie, naše zmeny v konfigurácii by mali byť vykonané v *local.ini*. Pretože chceme CouchDB púšťať na lokálnom stroji, ku ktorému sa pripojíme napríklad cez WiFi hotspot z mobilného telefónu, je potrebné CouchDB nastaviť aby používala lokálnu IP adresu zariadenia. Ďalej je potrebné nastaviť port, na ktorom bude CouchDB bežať. V konfigurácii je tiež možné nastaviť používanie SSL certifikátu, ten ale pre demonštráciu aplikácie nie je potrebný a nebude ani častou tejto práce. Počas vývoja sa môže tiež hodiť vlastné nastavenie administrátorského prístupu k databázam.

```
[httpd]
port = 5984   ;set port to run CouchDB on
bind_address = 0.0.0.0    ;bind CouchDB server to local address
[admins]
admin = mysecretpassword  ;set administrator account
```

Výpis 8.2: Minimálne nastavenie CouchDB konfigurácie

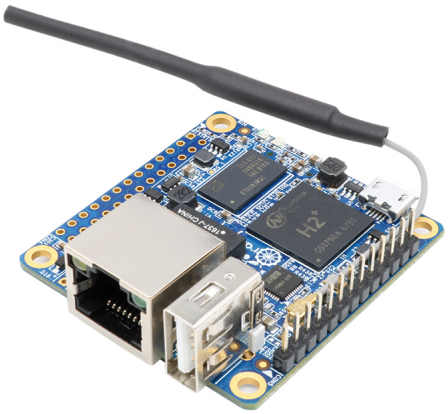
Vo svete CouchDB sa môžu zdať niektoré pravidlá z SQL konceptov porušované. Na tom nie je nič zlé a to vďaka tomu, že CouchDB je NoSQL databáza, ktorej koncepty sa dajú rôzne upraviť podľa potreby implementácie. Ak by bola CouchDB naďalej používaná aj v ostrej prevádzke v našej aplikácii, je úplne v poriadku ak by sme zvolili prístup, vytvárania samostatných databáz pre každú instanciu prevádzky. Takáto architektúra je vhodná, a napríklad CouchDB vo svojej dokumentácii odporúča aj vytváranie jedinečných databáz pre každého užívateľa aplikácie, ak je to potrebné.

Nasadenie CouchDB na ARM architektúru

Pretože sa snažíme postaviť riešenie s nízkou cenovkou pre užívateľa, mali by sme držať aj cenu serveru nízko. Pre našu aplikáciu by sa hodilo použitie mini počíatačov

ako napríklad Raspberry Pi. Na trhu je aj mnoho lacnejších alternatív, kde väčšina týchto mini počítačov je postavená na ARM architektúre procesorov.

Pre vývoj bola zakúpená doska Orange Pi Zero H2+. Je to mini počítač so 4-jadrovým procesorom ARM architektúry, 512MB RAM a 8GB eMMC pamäťou. Pre jej kúpu sme sa rozhodli práve z dôvodu dostatočného výkonu a dostupnej vnútornej eMMC pamäte, kde narozdiel od Rapberry Pi nie je potreba používania microSD kariet ako úložisko, čo nie je vhodné pre produkciu. Orange Pi Zero H2+ je predávaná už aj s WiFi anténou, takže užívatelia môžu používať riešenie v rámci prevádzky pripojením na WiFi AP. Osadený LAN port môžeme tiež využiť na premostenie Internetu, a zákazníci tak môžu na AP pristupovať aj k Internetu. Za cenu okolo $10 je toto serverové riešenie bezkonkurenčné.



Obr. 8.1: Mini počítač Orange Pi Zero H2+ [25]

CouchDB je síce možné nainštalovať na Linux OS pomocou utility *apt*, oficiálne repository ale obsahuje iba skompilované verzie CouchDB pre architektúry procesorov i386 (32-bitová architektúra) a AMD64 (x64-bitová architektúra). Aby sme mohli CouchDB prevádzkovať na ARM architektúre, je potrebné ju skompilovať na danej architektúre. Nižšie je ukážka inštalácie CouchDB ktorú sme previedli na mini počítači Orange Pi Zero H2+ s Linux OS. Inštaláciu sme prevádzali prostredníctvom pripojenia cez SSH, ovládaného z iného PC z dôvodu, že sme nemali k doske dostupný kit pre grafický čip a zobrazenie na monitore. Preto uvedieme kompletný postup tak, aby bol realizovateľný z utility príkazového riadku.

```
# install dependencies
sudo apt-get --no-install-recommends -y install \
  build-essential pkg-config erlang \
  libicu-dev libmozjs185-dev libcurl4-openssl-dev
```

```
# download source code of CouchDB
wget http:/ /mirror.dkm.cz/apache/couchdb/source/2.1.1/apache-
    couchdb-2.1.1.tar.gz

# unpack the source code
tar -xvzf apache-couchdb-2.1.1.tar.gz
cd apache-couchdb-2.1.1/

# if are all dependencies satisfied, run configure script
./configure

# build CouchDB
make release

# after build is complete, we can run CouchDB
sudo -i -u couchdb /home/couchdb/bin/couchdb
```

Výpis 8.3: Inštalácia CouchDB zo zdrojového kódu pomocou shell[19]

Po úspešnom skompilovaní a nasadení CouchDB na mini počítač ešte vytvoríme WiFi hotspot pre zariadenia na ktorých bude nainštalovaná naša aplikácia. V prípade, že bude doska pripojená k Internetu pomocou LAN, bude toto Internetové pripojenie prístupné aj na nami vytvorenom WiFi AP.

```
nmcli device wifi hotspot con-name HOTSPOT_NAME \
  ssid HOTSPOT_NAME \
  band bg \
  password HOTSPOT_PASSWORD
```

Výpis 8.4: Nastavenie WiFi hotspotu pomocou shell

Následne je ešte vhodné nastaviť nejakú službu, ktorá bude štartovať hotspot aj CouchDB službu po štarte. Na Internete je známych mnoho návodov, preto tu nebudeme tento postup ukazovať. Inou možnosťou tiež je, vytvoriť skript, ktorý sa bude automaticky spúšťať ihneď po štarte systému, čo je na systéme Linux tiež pomerne jednoduchá vec.

9 Test systému

V tejto časti bude venovaný priestor testovaniu systému pre podporu predaja. V prvej časti budú overené používateľské požiadávky čo ukáže úroveň splnenia zadania. V ďalšej časti bude prebiehať testovanie systému, ktoré má za úlohu zaručiť kvalitu riešenia.

Absencia akýchkoľvek automatických testov, unit testov alebo tiež automatických UI testov je bohužiaľ bežnou praxou aj pri zákazkovom vývoji. Obvykle je ich absencia spôsobená nereálnymi očakávaniami zákazníka na dobu vývoja, poprípade ak sa zákazník dozvie cenu vývoja automatizovaných testov stráca záujem. Pre vývoj našej aplikácie neboli písané automatizované testy z dôvodu nedostatku času. Použitie automatizovaných testov môže mnohokrát urýchliť vývoj, keďže praktickým opakom je, že vývojár po úprave kódu ručne preklikáva aplikáciu a testuje zmenu. Dovolíme si tvrdiť, že v našom prípade overovania konceptu by použitie automatizovaných testov veľa výhod neprinieslo, resp. ich vývoj by nebol rentabilný. Nedá sa odoprieť rýchlosť testovania automatizovanými testami, ale na druhej strane reálny kontakt užívateľa(vývojára) s aplikáciou pomôže odhaliť ďaleko viac možných chýb než úzko použiteľné automatické testy.

9.1 Overenie užívateľských požiadávok

Overenie splnenia všetkých požiadávok od zákazníka je veľmi dôležité, z pohľadu spracovateľa objednávky. V tejto časti sa overí splnenie užívateľských požiadávok na systém. Užívateľské požiadávky boli popísané v časti 3.1 formou User Stories. Táto časť zároveň slúži ako zhrnutie stavu aplikácie, ktoré poslúži ako referencia k ďalšiemu testovaniu. Je esenciálne pre testovanie, vedieť v akom stave je aplikácia, a aké funkcie môže od nej tester očakávať. Nasledujúci zoznam ukazuje úroveň implementácie aplikácie z pohľadu užívateľských požiadávok:

- **US1 Položky** - Táto požiadávka bola implementovaná formou karty *Položky*, na hlavnej obrazovke aplikácie. Užívateľ má možnosť upravovať tieto predajné položky, ako položky v zozname. Je možnosť novú položku pridať, upraviť, alebo vymyzať. Zoznam je rozbaľovací, očakáva sa členenie položiek do kategórií.
- **US2 Miesta** - Požiadávka bola implementovaná formou karty *Miesta*, na hlavnej obrazovke aplikácie. Užívateľ má možnosť upravovať tieto miesta, ako položky v zozname. Je možnosť nové miesto pridať, upraviť, alebo vymazať.
- **US3 Predaj** - Implementované v plnej miere, užívateľ môže vytvárať objednávky z karty *Miesta*, na hlavnej obrazovke, alebo v detaile daného miesta. Následne je možné objednávku zaplatiť v detaile miesta.

- **US4 Objednávky** - Za účelom tejto požiadávky bola implementovaná karta *Objednávky*, na hlavnej obrazovke. Užívateľom sa zobrazuje prehľad všetkých vytvorených objednávok, zoradený v čase.
- **US5 Detail zákazníka** - Požiadávka bola implementovaná. Užívateľ má možnosť rozkliknúť miesto v zozname *Miesta*, čím sa dostane do jeho detailu. Momentálne sa užívateľovi ukazujú základné údaje o zákaznikovi, ako jeho čas strávený na mieste, čas od poslednej objednávky a počet objednávok. Zároveň je v detaile jasne zobrazený zoznam objednaných položiek s cenami. a rovnako aj celková cena všetkých objednávok zákazníka.
- **US6 Synchronizácia** - Požiadávka bola implementovaná formou synchronizácie cez server. Server je možné nasadiť na samostatný vzdialený dedikovaný server, alebo na lokálny server, aby boli podporované aj prevádzky bez dostupnosti Internetového pripojenia. Implementácia synchronizácie je momentálne v stave overenia konceptu a nie je hodná na bežné používanie. Synchronizácia prebieha v reálnom čase, čo znamená, že ak na jednom zariadení v prevádzke spravíme zmenu, tá sa prejaví aj na ostatné zariadenia v primeranom čase. V prípade že bude zariadenie offline, alebo príde nové zariadenie do prevádzky, zreplikujú sa dáta do nového zariadenia tak, aby bol stav na klientských aplikáciach rovnaký.
- **US7 Štatistiky predajov** - Požiadávka bola implementovaná v plnej miere, pomocou webového zobrazenia štatistík v aplikácii. Momentálne štatistiky zobrazujú základné grafy a údaje o predajoch. Z dát ktoré aplikácia ukladá, je možné ich kombináciou a rôznymi algoritmami vypočítať ďalšie zaujímavé štatistické údaje.
- **US8 Štatistiky zamestnancov** - Požiadávka nebola implementovaná z dôvodu, že v aplikácii ešte nie je implementovaná správa užívateľov. Štatistiky zamestnancov by sa však mali zobrazovať v tom samom rozhraní ako štatistiky predajov, takže prakticky by nemalo byť náročné štatistické údaje o zamestnancoch doimplementovať. Je nutné podotknúť, že požiadávka je nad rámec zadania tejto práce.
- **US9 Vytvorenie prevádzky** - Požiadávka nebola implementovaná z časových dôvodov. Momentálne sa v aplikácii nijako nepoužíva dátový objekt prevádzky, no aktuálna implementácia dovoľuje jej pridanie a je v pláne pridať túto funkciu, esenciálnu pre používanie v reálnom prostredí. Požiadávka je nad rámec zadania tejto práce.
- **US10 Pozvanie do prevádzky** - Požiadávka nebola implementovaná z dôvodu, že v aplikácii nie je implementované používanie prevádzok *US9 Vytvorenie prevádzky*. Momentálne je užívateľom prevádzky ktokoľvek, kto si nainštaluje aplikáciu. Aplikácia je totiž v štádiu overenia konceptu a adresa serveru

pre pripojenie je nadefinovaná natvrdo v zdrojovom kóde.

- **US11 Doklad o kúpe** - Požiadávka nie je implementovaná z dôvodu časového deficitu. Počíta sa s implementáciou v budúcnosti, no aktuálne je požiadávka nad rámec zadania tejto diplomovej práce.

Vyššie uvedeným výpisom bola overená úroveň implementácie riešenia. Každopádne aplikácia pre podporu predaja nie je zatiaľ vhodná pre použitie v reálnom svete. Je potrebné najprv overiť, že prístup a zvolená architektúra je správna. V produkcii by bola cena opráv zásadne vyššia, než v štádiu vývoja.

9.2 Zaručenie kvality

9.2.1 Smoke testy

Smoke testy sa využívajú v dobe, keď je ukočená daná doba vývoja a je potreba overiť, že aplikácia funguje. Jedná sa o krátky test, ktorý overuje stabilitu verzie, a je ho nutné urobiť pred hĺbkovým systémovým testovaním. Tým sa eliminuje riziko, že testovanie fatálne zlyhá v dobe, keď systém už testuje väčšie množstvo testerov. Tým sa snažíme vyhnúť nárastu ceny testovania, v prípade chybnej verzie. U menších projektov, kde je testovanie zanedbávané, alebo nie je príliš organizované, sa zvyknú prevádzať len smoke testy.

Dielčie testy aplikácie boli vykonávané počas celej doby jej vývoja. Po implementovaní a otestovaní dielčej funkcie, boli následne otestované ostatné časti aplikácie, aby sme vylúčili možné chyby už v zárodku. Podobne tomu je aj teraz, pred začatím rozsiahlejšieho testovania, je potrebné vykonať základné testy, že aplikácia funguje správne. Pri testoch sme sa zamerali len na hlavné funkcie programu, ako je preklikanie obrazoviek, pridanie položiek do zoznamov a vytvorenie objednávky. Následne sme objednávku ukončili a sledovali štatistiky. Celý test prebiehal pri pripojení na server, pričom sme sledovali replikáciu dát na druhé zariadenie. Výsledok testu bol uspokojivý a neboli nájdené zásadné nedostatky. Aplikácia reagovala svižne a dáta sa prenášali aj na druhé zariadenie. Testovací scenár smoke testu ktorý sme prevádzali bol nasledovný:

1. Pridaj nové miesto predaja do aplikácie.
2. Pridaj novú kategóriu do položiek a napĺň ju ľubovoľnými položkami.
3. Vytvor novú objednávku.
4. Skontroluj, že sa objednávka správne vytvorila na obrazovke objednávok.
5. Skontroluj, že sa objednávka správne zobrazuje v detaile daného miesta.
6. Skontroluj, že sa objednávka reflektovala do štatistík predaja.
7. Zaplať objednávku.
8. Skontroluj, stav objednávky na obrazovke objednávok a detaile miesta.

Pri väčšej aplikácii by bolo vhodné prevádzať hĺbkové systémové testovanie podľa podrobne popísaných testovacích scenárov. Plné pokrytie aplikácie testovacími scenármi by však vyžadovalo veľké množstvo testovacích scenárov, čo nebolo naším zámerom. Naopak, v prípade malej aplikácie je rentabilnejšie udržiavať a testovať jednoduché scenáre smoke testov. Uvedeným jednoduchým testom sme otestovali hlavnú činnosť aplikácie.

9.2.2 Test aplikácie na rôznych zariadeniach

Aplikácia vyvýjaná pre mobilné platformy musí byť pred uvedením do produkcie testovaná na viacerých zariadeniach. Vzhľadom k testovaniu aplikácií na reálnych zariadeniach, je potreba otestovať rozličné operačné systémy, aj ich rôzne verzie. Dôležité je tiež otestovať rozličné rozlíšenia obrazoviek. Najčastejšie chyby ktoré prichádzajú z produkcie (okrem tých v logike aplikácie), sú chyby spojené s rozličnými operačnými systémami, alebo rozbitým grafickým rozhraním kvôli rôzným rozlíšeniam displejov zariadení.

Na platforme iOS je situácia jednoduchšia, kvôli uzavretému ekosystému firmy Apple. Aktuálne v roku 2018 Apple udržiava podporu pre zariadenia od roku 2015 a to menovite zariadenia: iPhone 6S/6S Plus, iPhone SE, iPhone 7/7 Plus, iPhone 8/8 Plus, iPhone X.[32] Ak to požiadávky aplikácie inak špecificky nevyžadujú, je dostačujúce testovať a smerovať vývoj aplikácie na tieto zariadenia. Reálne sa tieto mobilné zariadenia až tak nerozlišujú a v praxi sa zvykne testovať na jednom z menších displejov a na jednej verzii Plus s väčším displejom. Verzovanie systému iOS nezvykne byť zdrojom chýb pri vývoji aplikácií.

Na platforme Android je testovanie aplikácií náročnejšie. Zložitosť vychádza hlavne z rôznorodosti výrobcov, a potreby podporovať nespočetné množstvo zariadení, a rozlíšení displejov. Verzie systému Android podporu ale celkom uľahčujú, aplikácia sa nemôže správať rozlične v rámci jednej verzie Androidu, aj keď na zariadeniach rozličných výrobcov. V praxi nastávajú výnimky kedy to tak nie je, no sú veľmi ojedinelé a nemá zmysel sa nimi predbežne zaoberať. Programátor počíta s tým, že v rámci jednej verzie je zaručená plná kompatibilita a problémy rieši keď sa vyskytnú.

Aplikácia bola vyvinutá pomocou frameworku React Native bez použitia komponent pre špecifickú platformu, alebo zložitých knižníc závislých na hardvéri. Z toho dôvodu by aplikácia mala bežať na platforme iOS a Android bez väčších problémov. Pre testovanie aplikácie boli použité dostupné Android zariadenia:

- **Google Pixel 2** ako predstaviteľa vyššej triedy s verziou Androidu 8.1 a rozlíšením displeja 1080 x 1920 pixelov.

- **Meizu M2 Note** ako predstaviteľa strednej triedy s verziou Androidu 5.1 a rozlíšením displeja 1080 x 1920 pixelov.
- **Huawei Y5** ako predstaviteľa nižšej triedy s verziou Androidu 5 a rozlíšením displeja 480 x 854 pixelov.

Od testu nebolo očakávanie že odhalí veľké chyby, ale práve naopak sa dalo čakať že aplikácia bude bežať na všetkých zariadeniach bez chýb. Ako je vidieť zo špecifikácie, rozdiel v rozlíšení displeja zariadenia nižšej triedy, je pri teste dbať na správne zobrazenie prvkov, a či vplyvom rozlíšenia displeja nedochádza k rozbitiu grafického rozhrania aplikácie.

Na všetkých zariadeniach prebehol test s uspokojivým výsledkom. Priebeh testu bolo naštartovanie aplikácie a jednoduché preklikanie cez obrazovky a otestovanie implementovaných používateľských požiadaviek spísaných v časti 9.1. Test neodhalil žiadne chyby v logike aplikácie, alebo v jej nestabilite. Napriek tomu, podľa očakávaní, odhalil chyby v grafickom rozhraní. Jednou chybou, pomerne náročnou na odhalenie, bola rozbitá ovládacia časť na obrazovke štatistík. Zdalo sa, že je táto chyba spôsobená práve rozlíšením displeja, síce bola aplikácia napísaná tak, aby podporovala rôzne rozlíšenia. Ukázalo sa ale, že táto chyba bola spôsobená tým, že staršia verzia Androidu používala staršiu verziu komponenty WebView z jej SDK, ktorá nepodporovala niektoré nastavenia moderného *flex* CSS štýlovania. Chyba bola napokon odstránená použitím iného štýlovania.

Ďalšou bádateľnou chybou bola pomalšia odozva zariadenia na vykresľovanie animácií grafov v štatistikách. Animácia sa na pomalších zariadeniach javí ako pomalá a zasekaná. Táto chyba nebola odstránená v rámci práce, nakoľko nespôsobuje žiadne obmedzenie v používaní aplikácie. Do budúcna by sa hodilo obmedziť animáciu grafov iba na práve viditeľný graf, to by mohlo vyriešiť problém s nedostatočným výkonom. Poprípade úplne vypnúť animácie na menej výkonných zariadeniach.

9.2.3 Scenáre reálneho sveta

Program musí byť stabilný v prípade rôznych situácií ktoré môžu nastať v reálnom svete. Tieto situácie obnášajú používanie v offline režime, straty Internetového pripojenia, nečakané ukončenie aplikácie a ďalšie situácie, ktoré môžu nastať počas bežného užívania aplikácie. Nie je možné otestovať všetky scenáre, no je vhodné otestovať tie, ktoré s najväčšou pravdepodobnosťou nastanú.

Nečakané ukončenie aplikácie

Aplikácia môže byť v reálnej prevádzke kedykoľvek tvrdo vypnutá a musí sa s tým vysporiadať. Nami vytvorená aplikácia by nemala mať žiadne problémy v prípade

nečakaného ukončenia, čo potvrdili aj testy. Počas testu sme násilne ukončili aplikáciu, alebo vypli zariadenie. Problémovým bodom môže byť lokálna databáza ako perzistentná vrstva, ktorá nemusí znášať scenáre s tvrdým ukončením. Situácia by však mohla nastať iba v prípade tvrdého ukončenia počas zápisu. Pravdepodobnosť, že sa taká situácia stane je veľmi malá, no ošetriť takýto problém je zložité, ak nie nemožné. Očakávame, že použitá SQLite databáza je navhrnutá tak, že počíta s takýmito prípadmi, a v najhoršom prípade môže dôjst len k strate aktuálne zapisovaných dát, ale nie poškodeniu všetkých dát.

Synchronizácia

Aplikácia bola vyvinutá offline-first prístupom, musí teda fungovať v prípade straty spojenia so serverom. Po pripojení späť na server sa musia dáta správne zosynchronizovať, a aplikácia musí fungovať bez problémov ďalej. Testovanie ukázalo, že synchronizácia medzi zariadeniami funguje správne v jednoduchých scenároch, kedy sa zápisy dejú sekvenčne na rôzných zariadeniach. Rovnako v prípade, že je jeden klient offline, a zatiaľ sa dejú zmeny na dátach. Následne po pripojení si klient natiahne nové dáta a aplikácia funguje správne aj naďalej. Rovnako dobre funguje synchronizácia naopak, ak offline klient robí zmeny a pripojí sa, tie sa reflektujú do systému a na ostatné zariadenia.

Problém ale nastáva v prípade, ak sú dva zariadenia offline a obidva robia zmeny na dátach. Následne sa pripoja a to vytvorí konflikt pri replikácií dát. Keďže v našom systéme tabuľka predstavuje jeden CouchDB dokument, takže napríklad predávateľné položky sú uložené ako Json v jednom dokumente, tu nastáva problém. CouchDB si vyberie víťaza s novšou revíziou, a druhú revíziu zahodí(uloží mimo revízny strom). Tento scenár môže spôsobovať stratu dát a môže nastávať v reálnych podmienkach. Problém by bolo možné vyriešiť obmedzením používania aplikácie v offline stave, ale to nie je správne riešenie. Naopak, problém je treba vyriešiť bez obmedzenia užívateľa. CouchDB počíta s možnosťou riešenia konfliktov, a je možné ručne vyriešiť vzniknuté konflikty, čo ale tiež nie je vhodné a riešenie by malo byť automatizované. Našťastie CouchDB odošle na klientov v prípade konfliktu správu, takisto je možné sa spätne dotazovať na konflikty. Riešením je jednoduchá funkcia na strane PouchDB, ktorá v prípade konfliktu zmerguje dva konfliktné dokumenty, takže nedôjde k žiadnej strate dát.[33] Príklad takéhoto algoritmu by vyzeral napríklad takto:

1. Vráť mi revíziu dokumentu ktorá vyhrala.
2. Vráť mi list všetkých konfliktných dokumentov.
3. Vymaž víťaznú revíziu z listu.
4. Prejdi konfliktné dokumenty a skopíruj ich Json objekty.

5. Vymaž všetky revízie ktoré prehrali.
6. Žapíš novú víťaznú revíziu.

Konflikty pri synchronizácií sú bežným problémom a ostatné offline-first podporujúce frameworky ako Realm alebo Firebase riešia konflikty použitím verzie dát s najnovšou zmenou. Problém s konfliktami nebol vyriešený v rámci práce, no v prípade zanechania CouchDB a PouchDB je možné implementovať spomenutú opravu vo forme mergovacej funkcie.

10 Záver

V tejto práci bol vypracovaný návrh systému pre podporu predaja a jeho overenie následnou realizáciou. Po prieskume trhu sa ukázalo, že je možné ponúknuť zaujímave funkcie, a v prípade vývoja aplikácie pre široký rozsah užívateľov aj udržať cenu riešenia nízko. Bol navrhnutý koncept systému a takisto aj architektúra systému, kde boli zohľadnené všetky požiadávky na takýto systém. Následnou implementáciou sme sa pokúsili o overenie navrhnutého konceptu aplikácie. V poslednej časti práce sme sa venovali testovaniu vytvoreného riešenia.

Prieskum trhu nám ukázal, že by o takéto riešenie bol záujem. Záujem sme overili na vzorke potenciálnych užívateľov. Vytvorenie lacného riešenia systému pre podporu predaja, ktoré ponúka pokročilejšie funkcie vo forme štatistík má zmysel. Riešenie tiež ponúka istú modularitu a bude možné do budúcna pridávať ďalšie funkcie.

Po rozsiahlej analýze a návrhu riešenia, sme pomocu frameworku React Native implementovali cross-platformovú aplikáciu, pre najpoužívanejšie mobilné platformy iOS a Android. Počas celého návrhu aplikácie a jej implementácie sme dbali na udržanie vysokého UX pre užívateľov aplikácie. Návrhy ktoré sme vytvorili kladú dôraz na intuitívnosť riešenia.

V poslednej časti práce sme potom toto riešenie otestovali, čo ukázalo mieru splnenia zadania. Aplikáciu sme podrobili smoke testovaniu a testom na rôznych zariadeniach. Aplikácia prešla naším testovaním s uspokojivým výsledkom. Testovanie tiež odhalilo nedostatky, na ktoré sme navhrli možné riešenia.

Podarilo sa nám vytvoriť systém pre podporu predaja pre reštaurácie a iné prevádzky. Užívatelia majú možnosť v aplikácii spravovať miesta predaja, a svoje predajné položky. Používanie aplikácie uľahčí predaj na prevádzke, a ponúka prehľad o zákazníkoch, a ich objednávkach. Užívatelia majú tiež možnosť spätného prezerania trendov predaja. Zároveň môžu aplikáciu využívať spoločne s kolegmi, s automatickou synchronizáciou dát medzi zariadeniami.

Vytvorená aplikácia zatiaľ nevyhovuje systému finančnej správy Slovenskej a Českej republiky ako registračná pokladnica, no môže byť použitá pre podporu predaja. Aj keď aplikácia nie je zatiaľ vhodná používania v reálnom svete, overili sme koncept jej realizácie. Ako slabú stránku vidíme použitie PouchDB a CouchDB databáz, nakoľko nad nimi a ich synchronizáciou nemáme plnú kontrolu. Do budúcna by bolo vhodné dokončenie implementácie vlastného riešenia pre synchronizáciu dát. Tiež by bolo vhodné implementovať protokol pre komunikáciu s tlačiarňou pokladničných blokov, aby mohla byť aplikácia použiteľná ako hlavný pokladničný systém v prevádzkach.

Literatúra

[1] Beck, K., Beedle, M., Bennekum, A. V., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D. *Principles behind the Agile Manifesto* [online]. [cit. 2018-4-9]. Dostupné z: `http://agilemanifesto.org/iso/en/principles.html`

[2] Trello lets you work more collaboratively and get more done [online]. [cit. 2018-4-9]. Dostupné z: `https://trello.com/`

[3] Zlaté stránky: online databáza firiem [online]. [cit. 2018-4-9]. Dostupné z: `https://www.zlatestranky.sk/hladanie/restauracie/`

[4] Formuláre Google: zadarmo vytvárajte a analyzujte prieskumy [online]. [cit. 2018-4-9]. Dostupné z: `https://docs.google.com/forms/`

[5] Gartner *Worldwide Smartphone Sales to End Users by Operating System in 2017* [online]. [cit. 2018-4-9]. Dostupné z: `https://www.gartner.com/newsroom/id/3859963`

[6] Leszczynski M. *GetResponse - Email Marketing Benchmarks* [online]. [cit. 2018-4-9]. Dostupné z: `https://www.getresponse.com/resources/reports/email-marketing-benchmarks.html`

[7] Elcom Cash Registers [online]. [cit. 2017-12-24]. Dostupné z: `https://www.elcom.eu/`

[8] Loyverse Point of Sale System [online]. [cit. 2017-12-24]. Dostupné z: `https://loyverse.com/`

[9] Fiscal memory device. In: *Wikipedia: the free encyclopedia* [online]. [cit. 2017-12-24]. Dostupné z: `https://en.wikipedia.org/wiki/Fiscal_memory_devices`

[10] Zbierka zákonov SR *Zákon č.289/2008 Z. z. o používaní elektronickej registračnej pokladnice a o zmene a doplnení zákona Slovenskej národnej rady č.511/1992 Zb. o správe daní a poplatkov a o zmenách v sústave územných finančných orgánov v znení neskorších predpisov* [online]. [cit. 2017-12-24]. Dostupné z:
https://www.financnasprava.sk/_img/pfsedit/Dokumenty_PFS/Zverejnovanie_dok/Sprievodca/Sprievodca_danami/2017.07.28_Spriev_dan.pdf

[11] Etržby: elektronická evidence tržeb [online]. [cit. 2017-12-24]. Dostupné z:
http://www.etrzby.cz/

[12] 1and1 Digital Guide *Legal requirements for cash registers* [online]. [cit. 2017-12-24]. Dostupné z:
https://www.1and1.com/digitalguide/small-business/taxes-law/points-of-sale-a-guide-to-cash-registers/

[13] Internal Revenue Service *What kind of records should I keep?* [online]. [cit. 2017-12-24]. Dostupné z:
https://www.irs.gov/businesses/small-businesses-self-employed/what-kind-of-records-should-i-keep

[14] Census: U.S. and World Population Clock [online]. [cit. 2017-12-24]. Dostupné z:
https://www.census.gov/popclock/

[15] Realm: Create reactive mobile apps in a fraction of the time [online]. [cit. 2017-12-24]. Dostupné z:
https://realm.io/

[16] React Native: Build native mobile apps using JavaScript and React [online]. [cit. 2017-12-24]. Dostupné z:
https://facebook.github.io/react-native/

[17] Redux State Container documentation [online]. [cit. 2017-12-24]. Dostupné z:
https://redux.js.org/

[18] PouchDB, the Javascript Database that Syncs! [online]. [cit. 2017-12-24]. Dostupné z:
https://pouchdb.com/

[19] Apache CouchDB [online]. [cit. 2017-12-24]. Dostupné z:
http://couchdb.apache.org/

[20] Gilbert S., Lynch N. *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*. ACM SIGACT News, Jun 2002, vol. 33, no. 2.

[21] Mehra A. *Understanding the CAP Theorem* [online]. [cit. 2018-04-28]. Dostupné z:
`https://dzone.com/articles/understanding-the-cap-theorem`

[22] MARTIN, Robert C. *Clean code: a handbook of agile software craftsmanship. Upper Saddle River*. NJ: Prentice Hall, c2009. ISBN 0132350882

[23] React Navigation: Routing and navigation for your React Native apps [online]. [cit. 2018-04-28]. Dostupné z:
`https://reactnavigation.org/`

[24] Redux Thunk: Thunk middleware for Redux [online]. [cit. 2018-04-28]. Dostupné z:
`https://github.com/gaearon/redux-thunk`

[25] Orange Pi Zero: manufacturer homepage [online]. [cit. 2018-04-28]. Dostupné z:
`http://www.orangepi.org/orangepizero/`

[26] Parse Platform: The open source version of the Parse backend [online]. [cit. 2018-04-28]. Dostupné z:
`http://parseplatform.org/`

[27] D3.js: Data-Driven Documents [online]. [cit. 2018-04-28]. Dostupné z:
`https://d3js.org/`

[28] Chart.js: Open source HTML5 Charts for your website [online]. [cit. 2018-04-28]. Dostupné z:
`https://www.chartjs.org/`

[29] Moment.js: Parse, validate, manipulate, and display dates and times in JavaScript [online]. [cit. 2018-04-28]. Dostupné z:
`https://momentjs.com/`

[30] w3schools: HTML DOM Events [online]. [cit. 2018-04-28]. Dostupné z:
`https://www.w3schools.com/jsref/dom_obj_event.asp`

[31] Blank G. *Github: React Native WebView bridge sample* [online]. [cit. 2018-04-28]. Dostupné z:
`https://github.com/blankg/rn-webview-bridge-sample`

[32] List of iOS devices. In: *Wikipedia: the free encyclopedia* [online]. [cit. 2018-04-28]. Dostupné z:
`https://en.wikipedia.org/wiki/List_of_iOS_devices`

[33] Bird G. *CouchDB conflict resolution sample code* [online]. [cit. 2018-04-28]. Dostupné z:
`https://github.com/glynnbird/deconflict`

Zoznam symbolov, veličín a skratiek

POS	predajné miesto – Point of Sale
OR	ratio otvorenia – Open Rate
CTR	ratio prekliknutia – Click-Through Rate
POC	overenie koncepcie – Proof of Concept
OS	operačný systém – Operating System
USB	univerzálna sériová zbernica – Universal Serial Bus
LAN	lokálna počítačová sieť – Local Area Network
AP	prístupový bod – Access Point
UX	užívateľský prežitok – User Experience
IRS	Služba interných príjmov – Internal Revenue Service
SDK	softvérový vývojový balíček – Software Development Kit
JVM	Java virtuálny stroj – Java Virtual Machine
ORM	objektovo-relačné mapovanie – Object-Relational Mapping
DAO	objekt pre prístup k dátam – Data access object
UI	užívateľské prostredie – User Interface
Json	JavaScriptový objektový zápis – JavaScript Object Notation
XML	rozšíriteľný načkovací jazyk – Extensible Markup Language
HTTP	hypertextový prenosový protokol – Hypertext Transfer Protocol
API	rozhranie pre programovanie aplikácií – Application Programming Interface
REST	reprezentovateľný stavový prenos – Representational State Transfer
CSS	kaskádové štýly – Cascading Style Sheets
DOM	objektový model dokumentu – Document Object Model
HTML	hypertextový značkový jazyk – Hypertext Markup Language